Satisfiabilité propositionnelle et raisonnement par contraintes : modèles et algorithmes

THÈSE

présentée et soutenue publiquement 6 Décembre 2011

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Jean-Marie Lagniez

Composition du jury

Président :	Éric Grégoire	Université d'Artois
Rapporteurs :	Arnaud Lallouet	Université de Caen - Basse Normandie
	Laurent Simon	Université Paris-Sud 11
Examinateurs :	Emmanuel Hebrard	LAAS Toulouse
	Chu Min Li	Université de Picardie
Examinateur invité :	Frédéric Lardeux	Université d'Angers
Co-Directeurs :	Gilles Audemard	Université d'Artois
	Bertrand Mazure	Université d'Artois
Directeur de Thèse :	Lakhdar Saïs	Université d'Artois

Centre de Recherche en Informatique de Lens – CNRS UMR 8188
Université d'Artois, rue Jean Souvraz, S.P. 18 F-62307, Lens Cedex France
Secrétariat : Tél.: +33 (0)3 21 79 17 23 – Fax : +33 (0)3 21 79 17 70
http://www.cril.univ-artois.fr

Mise en page avec memcril (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Remerciements

Les remerciements.

À mes frères.

Table des matières

II Méthodes de résolution hybride

Liste des tableaux

Table des figures

Liste des Algorithmes

Introduction générale

Avec l'avènement des ordinateurs, nous sommes aujourd'hui amenés à manipuler des données d'une complexité croissante et de plus en plus nombreuses. Un programme peut aisément atteindre plusieurs centaines de milliers de lignes, les processeurs se miniaturisent mais effectuent des opérations de plus en plus complexes, les compagnies aériennes veulent remplir au mieux leur avions, *etc.* De tels exemples existent dans à peu près tous les domaines possibles et imaginables. Dès lors, la vérification des programmes, des micro-processeurs ou encore l'allocation au mieux des sièges des compagnies aériennes ne peuvent plus être réalisées par des humains mais doivent être automatisés. Cette automatisation nécessite une modélisation du problème initial, puis un algorithme permettant de résoudre le problème ainsi modélisé. Une alternative à l'élaboration d'algorithmes dédiés à un type de problème consiste à utiliser la programmation par contraintes. En effet, elle constitue un paradigme de résolution très puissant permettant de modéliser de nombreux problèmes (dont les exemples cités ci-dessus). Le problème SAT, pour « *SATisfiability* » et le problème CSP, pour « *Constraint Satisfaction Problems* » sont tous les deux des thèmes majeurs de la programmation par contraintes. Le principal avantage de ces paradigmes est la séparation entre la partie modélisation et la partie résolution. Une fois le problème modélisé, il est possible d'utiliser un des nombreux solveurs existant dans la communauté pour le résoudre. C'est ainsi que, depuis plusieurs années, des chercheurs se focalisent sur l'amélioration des techniques de résolution pour les problèmes SAT et CSP. Cette alternative a donné de très bons résultats dans la pratique, quelque fois meilleurs que des démonstrateurs ad-hoc.

La logique propositionnelle (Boole 1854) sur laquelle est basée le problème SAT est un formalisme de représentation des plus simplistes. Elle contient un nombre fini de variables qui ne peuvent prendre que deux valeurs de vérité (*vrai* et *faux*) et utilise les connecteurs logiques classiques (le *et*, le *ou* et enfin la négation *non*). Malgré sa simplicitié, le problème SAT, premier problème a avoir été démontré *NP-complet* (Cook 1971), est un problème central en théorie de la complexité, mais aussi en intelligence artificielle et en démonstration automatique. Cela fait maintenant une dizaine d'années, et l'avènement d'un nouveau type de solveurs (nommé CDCL « *Conflict Driven, Clause Learning* » (Moskewicz *et al.* 2001a)), que le problème SAT est utilisé dans la résolution de problèmes extrêmement divers. Le plus emblématique d'entre eux est la vérification formelle bornée (BMC « *Bounded Model Checking* » (Biere *et al.* 1999a)) puisqu'il a été parmi les premiers problèmes à être modélisé avec succès en SAT. Nous pouvons encore citer la gestion des dépendances dans les paquets d'une distribution linux (Argelich *et al.* 2010) ou encore dans les plugins du logiciel Eclipse (Le Berre et Rapicault 2009). En effet, à chaque fois qu'un plugin est installé ou mis à jour dans ce logiciel, un solveur SAT, nommé SAT4J (Le Berre et Parrain), est appelé afin de déterminer s'il n'y a pas de conflits et quels sont les autres plugins qui doivent également être installés. Ceci fait sûrement de SAT4J le solveur SAT le plus utilisé du monde ! Et puis, n'est-ce pas Edmund Clarke, prix Turing en 2007 avec Allen Emerson et Joseph Sifakis, pour leur travaux sur la vérification formelle, qui a dit : « *Clearly, efficient* SAT *solving is a key technology for 21st century computer science* » (Biere *et al.* 2009). Pour autant, de nombreux challenges existent encore dans la communauté. L'élaboration de nouvelles techniques incomplètes se focalisant sur les instances (ou problèmes) non satisfiables en est encore aux balbutiements. Les méthodes hybrides, combinant deux méthodes de recherche différentes, ne donnent toujours pas les résultats espérés. De plus, il ne faut pas considérer le problème SAT comme la seule possibilité et de nombreux problèmes restent encore inabordables pour les solveurs SAT.

Le problème CSP possède un formalisme beaucoup plus expressif que le problème SAT. Il utilise des variables qui peuvent prendre un nombre fini de valeurs. Il peut également coder les contraintes de manière intentionnelle ($3 \times X + 4 \times y \leq 4$, par exemple) mais également utiliser des contraintes globales

(la contrainte *alldif* étant la plus connue (Régin 1999)) utilisant des algorithmes dédiés de filtrage. Ainsi, certains problèmes sont inaccessibles aux solveurs SAT car ils nécessitent trop de variables, mais peuvent facilement être résolus à l'aide de solveurs CSP (citons par exemple les problèmes d'allocation de fréquences (FAPP)). Une extension de CSP, les CSP pondérés, permet en plus de résoudre des problèmes primordiaux en contrainte : les problèmes d'optimisation.

Les problèmes SAT et CSP sont donc aujourd'hui utilisés quotidiennement dans des domaines très variés, par conséquent les solveurs associés doivent résoudre des problèmes de plus en plus complexes, nécessitant l'élaboration constante de nouveaux algorithmes de résolution et l'amélioration de ceux déjà existants. Les progrès réalisés tous les deux ans dans les compétitions SAT témoignent de ces améliorations (PRECOSAT (Biere 2009) et GLUCOSE (Audemard et Simon 2009a), les deux vainqueurs de la compétition de 2009 auraient respectivement terminé 6ème et 14ème en 2011). Cette thèse s'inscrit principalement dans cette perspective. Nous nous sommes attelés à proposer de nouveaux algorithmes et à améliorer des méthodes existantes dans le cadre SAT. La proximité des formalismes SAT et CSP, qui a toujours permis des transferts de résultats entre ces deux domaines, nous a également amené à étendre certains de ces algorithmes au formalisme CSP. Nos contributions s'articulent donc autour de deux principaux points : d'une part, l'hybridation de différents types de solveurs et de l'autre, l'amélioration des solveurs de type CDCL.

Ce document est divisé en trois parties. Tout d'abord nous introduisons les notions essentielles à la compréhension de ce manuscrit : les bases de la théorie de la complexité, la définition formelle des problèmes SAT et CSP, ainsi que la notion de classes d'instances sont présentées. Un état de l'art concernant les différents paradigmes de résolution est également établi (algorithme de recherche locale, approche complète, méthode hybride et approche de résolution parallèle pour SAT).

La seconde partie est consacrée à nos contributions au cadre de la résolution d'un des challenges proposés par Selman *et al.* (1997), qui vise à montrer que la collaboration des deux principales méthodes de résolution (recherche locale et méthode complète) dépasse en efficacité chacune d'entre elles prises séparément. Notre contribution dans ce cadre est double : en effet, dans un premier temps, nous présentons un nouveau concept permettant d'étendre la notion d'analyse de conflits telle qu'elle est faite dans les solveurs SAT modernes dans le cadre de la recherche locale (Audemard *et al.* 2009a;b). Ces travaux ont abouti au développement d'un algorithme, nommé CDLS (*Conflict Driven Local Search*), qui est présenté et comparé expérimentalement aux meilleures méthodes connues. Les travaux conduits sur cette méthode nous ont amené à élaborer un nouveau schéma d'hybridation permettant de faire collaborer efficacement et de manière bidirectionnelle une méthode de recherche locale et un algorithme complet de type recherche en profondeur d'abord. Ce schéma a ensuite été appliqué et étudié expérimentalement dans le cadre de la résolution pratique des problèmes SAT (SATHYS) et CSP (FAC-SOLVER) (Audemard *et al.* 2009c; 2010a;c;b, Grégoire *et al.* 2011a;b).

Finalement, dans la troisième partie, nous proposons deux contributions s'intégrant dans le cadre de la résolution du problème SAT. Notre première contribution consiste en une nouvelle stratégie de réduction de la base de clauses apprises, utilisée classiquement dans les solveurs CDCL, basée sur l'activation et la désactivation de clauses au cours de la recherche (Audemard *et al.* 2011a;b). Ces travaux ont conduit à l'élaboration d'une nouvelle mesure, nommée PSM, permettant de prédire si une clause sera utile pour la suite de la recherche. La viabilité de ce concept est démontrée au travers d'un ensemble d'expérimentations. La seconde contribution s'intègre quant à elle dans le cadre de la résolution pratique de SAT à l'aide de la collaboration de plusieurs solveurs CDCL lancés en parallèle (Guo et Lagniez 2011a;b). Nous proposons, dans ce cadre, d'ajuster dynamiquement la configuration d'un des solveurs lorsqu'il est établi qu'un autre effectue un travail similaire. Pour cela, nous avons décrit une nouvelle mesure permettant d'estimer si deux unités de calcul explorent de la même manière l'espace de recherche développé.

Première partie
État de l’art

Introduction de l'étude bibliographique

La programmation par contraintes est un paradigme de résolution très puissant permettant de modéliser de nombreux problèmes issu de domaines variés, tels que les interfaces homme-machine, l'intelligence artificielle ou encore la recherche opérationnelle. Dans le cadre de cette thèse, nous nous intéressons à deux formalismes utilisés pour la modélisation de ce type de problèmes : la logique propositionnelle et les réseaux de contraintes. Ces derniers permettent de représenter un problème par un ensemble de variables prenant leur valeur dans un ensemble fini et liées par un ensemble de contraintes mathématiques ou symboliques. Bien que d'apparence rudimentaire, ces deux formalismes permettent de modéliser de nombreux problèmes réels tels que les problèmes de planification (Blum et Furst 1997, Do et Kambhampati 2001), de tournées de véhicules (Laburthe 1997, Pesant *et al.* 1998), de prédire la structure tridimensionnelle d'une protéine (Backofen et Will 2003), de *model checking* (Biere *et al.* 1999a), *etc.* L'utilisation de ces paradigmes permet ainsi de séparer la partie modélisation de la partie résolution. Résoudre efficacement un problème repose alors sur l'élaboration d'algorithmes puissants et consiste le plus souvent à savoir s'il existe une affectation des valeurs des variables permettant de satisfaire toutes les contraintes exprimées. Indiquer si une telle solution existe est appelée « problème de décision ». Les problèmes de satisfaisabilité d'une formule propositionnelle (SAT) et de satisfaction de contraintes (CSP) sont respectivement les deux problèmes de décision associés à la recherche d'une solution dans le cas d'une formule propositionnelle et d'un réseau de contraintes.

Cette partie se compose de cinq chapitres. Le premier chapitre est consacré à la présentation des problèmes SAT et CSP. Après avoir introduit quelques notions de complexité afin de situer au mieux ces deux problèmes et leur intérêt théorique, nous définissons les deux formalismes que sont la logique propositionnelle et les réseaux de contraintes. Une fois ces formalismes définis, nous donnons certaines notations, définitions et propriétés utiles à la compréhension de la suite de ce manuscrit.

Le second chapitre est consacré à la présentation de la recherche locale. Pour commencer, nous introduisons de manière générale les principales composantes utiles à l'élaboration de tels algorithmes (notion de voisinage, minimum local, fonction d'évaluation) ainsi que leur mise en œuvre pour les problèmes SAT et CSP. Nous présentons ensuite différentes stratégies d'échappement parmi les plus utilisées et une méthode pour régler dynamiquement les différents paramètres utilisés dans ce genre d'approche.

Le troisième chapitre présente les algorithmes complets de résolution et est divisé en deux parties. Dans la première, nous présentons les algorithmes complets dans le cadre de SAT. Plus précisément, nous détaillons les différentes méthodes et composantes qui ont conduit à l'élaboration des solveurs SAT modernes. La seconde partie est quant à elle consacrée à la présentation des algorithmes complets dans le cadre CSP. Dans cette dernière, nous nous intéressons plus particulièrement aux algorithmes de recherche avec retour arrière et aux différentes formes de cohérences.

Ensuite, nous consacrons un chapitre à la présentation d'algorithmes hybrides. Nous présentons dans un premier temps les différents schémas permettant de combiner la recherche locale et une méthode complète. Puis nous détaillons quelques approches basées sur ces derniers et donnons quelques exemples d'implémentation.

Cette partie se termine par un chapitre consacré à la résolution du problème SAT en parallèle. Après avoir rappelé l'enjeu qu'il y a à résoudre SAT de cette manière, nous introduisons rapidement le concept d'architecture multi-cœurs et les deux types d'approches utilisés afin de résoudre SAT en parallèle.

Chapitre 1

Problématique, Définitions et Notations

Sommaire

CE CHAPITRE a pour objectif de présenter les problèmes de satisfaction de contraintes SAT et CSP. Nous commençons par quelques notions de complexité afin de situer au mieux ces deux problèmes. Ensuite nous définissons les problèmes de la satisfaisabilité d'une formule propositionnelle (SAT) et d'un réseau de contraintes (CSP). Enfin nous consacrons la dernière section à la présentation des différents types de problèmes pouvant être encodés à l'aide de ces deux formalismes.

1.1 Introduction à la théorie de la complexité

Cette section a simplement pour objectif de situer les problèmes SAT et CSP dans la hiérarchie polynomiale, nous ne rappelons pas l'ensemble des concepts et propriétés nécessaires à la compréhension de la théorie de la complexité. Pour plus de précisions quant à ce domaine, le lecteur pourra se référer utilement aux ouvrages (Turing et Girard 1991, Papadimitriou 1994, Creignou *et al.* 2001).

La complexité d'un programme est une mesure des ressources nécessaires à son exécution. Les ressources qui sont prises en compte sont, usuellement, le temps et l'espace. La théorie de la complexité étudie les problèmes qui sont calculables avec une certaine quantité de ressources. Il ne faut pas confondre la complexité d'un problème avec sa calculabilité. En effet, la théorie de la complexité se concentre

uniquement sur les problèmes qui peuvent être résolus, la question étant de savoir s'ils peuvent être résolus efficacement ou pas en se basant sur une estimation (théorique) des ressources nécessaires. Plus précisément, l'objectif de cette théorie est d'étudier l'évolution des ressources nécessaires à la résolution d'un problème en fonction de la taille des données fournies en entrée. Autrement dit, étant donnée une procédure P, la théorie de la complexité s'intéresse à la variation de la durée ou l'espace nécessaire au calcul de $\mathsf{P}(n)$ en fonction de la taille de l'argument n.

De manière générale, le temps de calcul est considéré comme la ressource la plus significative pour l'évaluation de la complexité d'un algorithme, la mémoire consommée lui étant très liée. Ainsi dans la suite de ce manuscrit, lorsqu'il est fait mention de complexité et sauf mention contraire, il est admis que c'est de la complexité temporelle qu'il s'agit.

1.1.1 Notions de bases

L'évaluation de la complexité en temps des algorithmes repose sur une définition précise du temps qui ne dépend pas de la vitesse d'exécution de la machine ni de la qualité du code produit par le compilateur. En effet, le temps de calcul obtenu par l'exécution d'un programme sur une machine donnée ne reflète pas réellement la difficulté de celui-ci (un problème ne devient pas plus facile parce qu'un nouveau processeur est sur le marché). Afin d'estimer plus finement la complexité d'un algorithme, il est nécessaire de s'abstraire de cette notion temporelle dépendante du matériel et de considérer le temps comme le nombre d'opérations élémentaires $T_{\mathsf{P}}(n)$ nécessaires à l'exécution d'un programme pour une certaine donnée de taille n.

Exemple 1.1. *Considérons le problème qui consiste à décider si un élément e se trouve dans un tableau trié tab de n éléments. Pour réaliser cette recherche il existe plusieurs méthodes différentes. Parmi ces dernières, une approche naïve, que nous nommons P, consiste à parcourir les éléments dans l'ordre jusqu'à trouver l'élément recherché. Pour cette méthode, plusieurs cas peuvent se présenter :*

- *Le meilleur cas est celui où l'élément recherché se trouve dans la première case du tableau et dans ce cas $T_{\mathsf{P}}(n) = 1$;*
- *Le pire cas est celui où l'élément est le dernier du tableau et dans ce cas $T_{\mathsf{P}}(n) = n$;*
- *Le cas moyen dépend de la répartition probabiliste des éléments du tableau. Dans le cas d'une distribution uniforme, le nombre moyen d'opérations pour exécuter le problème est de $T_{\mathsf{P}}(n) = \frac{n}{2}$.*

En pratique, lorsqu'aucune précision n'est apportée, c'est souvent le comportement de l'algorithme dans le pire des cas qui est étudié. En effet, le nombre d'opérations effectuées dans le meilleur des cas n'apporte aucune information sur le comportement réel de l'algorithme. Quant à la valeur obtenue en moyenne, bien que permettant de révéler le comportement « réel » de l'algorithme, elle est difficile à calculer et dépend fortement d'hypothèses effectuées sur la distribution des données.

Définition 1.1 (Complexité algorithmique)**.** *Soit n la taille de la donnée en entrée. Un programme P est de complexité $\mathcal{O}(f(n))$ dans le pire des cas s'il existe une certaine constante $c < \infty$ tel que :*

$$\lim_{n\to\infty} \frac{T_{\mathsf{P}}(n)}{f(n)} = c$$

Exemple 1.2. *L'algorithme proposé dans l'exemple 1.1 est de complexité $\mathcal{O}(n)$.*

Définition 1.2 (Algorithme polynomial)**.** *Soit P un algorithme exécuté sur une entrée de taille n. P est dit polynomial s'il existe un entier i tel que P est de complexité $\mathcal{O}(n^i)$, c'est-à-dire que le nombre d'opérations nécessaires à la terminaison de $\mathsf{P}(n)$ est majoré par un polynôme en la taille n de l'entrée.*

En général, la complexité n'est pas donnée de manière exacte. En pratique, seul un ordre de grandeur asymptotique est calculé. Par exemple, supposons que le nombre d'opérations nécessaires à la terminaisons d'un programme P sur une donnée de taille n soit précisément de $n^2 + 2n + 7$. Dans ces conditions, il est admis que le programme P est en $\mathcal{O}(n^2)$.

Définition 1.3 (Algorithme exponentiel)**.** *Soit* P *un algorithme exécuté sur une entrée de taille* n*,* P *est simplement exponentiel s'il existe un réel* τ *strictement supérieur à* 1 *et un polynôme* $f(n)$ *en* n *tels que* P *est de complexité* $\mathcal{O}(\tau^{f(n)})$.

Remarque 1.1. *Dans la suite de ce manuscrit, nous utilisons par souci de concision le terme exponentiel plutôt que simplement exponentiel.*

Du point de vue pratique, il y a une nette différence de temps de résolution entre les algorithmes polynômiaux et les algorithmes exponentiels. Afin de donner une idée de l'ordre de grandeur, nous reportons sur le tableau 1.1 les temps de calculs nécessaires à l'exécution de différents programmes en fonction de leur complexité.

Complexité	Type de Complexité	$n = 10$	$n = 100$	$n = 1000$
$\mathcal{O}(1)$	constante	$10^{-9}s$	$10^{-9}s$	$10^{-9}s$
$\mathcal{O}(log(n))$	logarithmique	$2.3 \times 10^{-9}s$	$4.6 \times 10^{-9}s$	$6.9 \times 10^{-9}s$
$\mathcal{O}(n)$	linéaire	$10^{-8}s$	$10^{-7}s$	$10^{-6}s$
$\mathcal{O}(nlog(n))$	linéarithmique	$2.3 \times 10^{-8}s$	$4.6 \times 10^{-7}s$	$6.9 \times 10^{-6}s$
$\mathcal{O}(n^2)$	quadratique	$10^{-7}s$	$10^{-5}s$	$10^{-3}s$
$\mathcal{O}(n^3)$	cubique	$10^{-6}s$	$10^{-3}s$	$1s$
$\mathcal{O}(2^n)$	exponentielle	$10^{-6}s$	$1.2 \times 10^{21}s$	$10^{300}s$
$\mathcal{O}(n!)$	factorielle	$3,6 \times 10^{-3}s$	$9.3 \times 10^{148}s$	$4 \times 10^{2558}s$

TABLE 1.1 – Évolution du temps de calcul en fonction de la complexité d'un algorithme et de la taille des données. Les temps d'exécution sont estimés sur la base de $10^{-9}s$ par instruction.

Comme nous pouvons l'observer sur le tableau 1.1, les différences de temps nécessaires à la résolution de problèmes avec des complexités différentes peuvent être astronomiques. Plus particulièrement les problèmes d'une complexité exponentielle (et plus) sont généralement impossibles à réaliser sur des données de taille raisonnable ($n > 100$). Compte tenu de cette particularité, deux questions s'imposent :

– existe-t-il des problèmes qui n'admettent pas d'algorithme polynomial ?
– si oui, est-il possible de déterminer qu'un problème n'admet pas d'algorithme polynomial ?

Ce sont à ces questions que tente de répondre la théorie de la complexité présentée dans la section suivante.

1.1.2 La machine de Turing

Les machines de Turing ne sont pas des machines matérielles destinées à être construites et à être utilisées en pratique. Elles fournissent un modèle théorique pour les ordinateurs et les algorithmes. Elles sont caractérisées par un dispositif qui est commun aux machines de Turing déterministes et aux machines de Turing non déterministes, et par une fonction de transition qui dépend de l'algorithme à représenter. Les machines de Turing évoluent par étapes successives, appelées « pas ». Formellement une telle machine se définit comme suit :

Définition 1.4 (Machine de Turing)**.** *La mise en œuvre concrète d'une machine de Turing est réalisée à l'aide des éléments suivants (voir figure 1.1 pour une illustration) :*

1. *Un « ruban » divisé en cases consécutives. Chaque case contient un symbole parmi un alphabet fini $\mathcal{S}$. L'alphabet contient un symbole spécial « blanc » (b), et un ou plusieurs autres symboles. Le ruban est supposé être de longueur infinie vers la gauche ou vers la droite, en d'autres termes la machine doit toujours avoir assez de longueur de ruban pour son exécution. Les cases non encore écrites du ruban contiennent le symbole « blanc » ;*
2. *Une « tête de lecture/écriture » qui permet de lire et d'écrire les symboles sur le ruban, et de se déplacer vers la gauche ou vers la droite du ruban ;*
3. *Un « registre d'état » $\mathcal{E}$ qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini et il existe un état spécial appelé « état de départ » qui est l'état initial de la machine avant son exécution ;*
4. *Une « fonction de transitions » t qui indique à la machine quel symbole écrire, comment déplacer la tête de lecture (G pour gauche, D pour droite), et quel est le nouvel état, en fonction du symbole lu sur le ruban et de l'état courant de la machine. Si aucune action n'existe pour une combinaison donnée d'un symbole lu et d'un état courant, la machine s'arrête.*

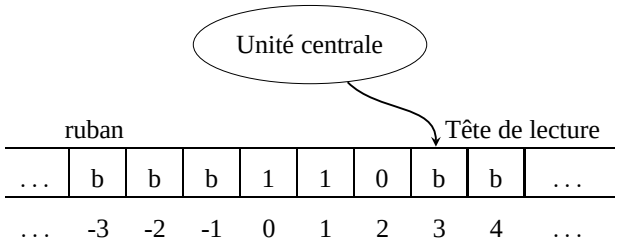


FIGURE 1.1 – Représentation schématique d'une machine de Turing. L'unité centrale contrôle l'ensemble du déroulement du programme et son contenu (fonction de transition) dépend du programme à exécuter.

Un calcul sur une machine de Turing s'effectue de la manière suivante :
– la donnée d'entrée est placée sur la bande, la tête de lecture est positionnée sur la première case de la donnée ;
– le programme se déroule suivant la fonction de transition et se termine lorsqu'une transition conduit à un état d'arrêt ;
– le résultat peut alors être lu sur la bande.

Remarque 1.2. *Il est possible que la machine n'atteigne jamais un état d'arrêt, il est dans ces conditions dit que la machine boucle.*

Il est à noter que l'ensemble de symboles $\mathcal{S}$, s'il reste « raisonnable », n'est pas primordial. Afin de fixer les idées, le lecteur peut provisoirement penser à un choix classique : $\mathcal{S} = \{0, 1, b\}$ (choix retenu pour la figure 1.1). La spécificité d'une machine de Turing se fait principalement en précisant l'ensemble $\mathcal{E}$ des états et la fonction de transitions t. Un « pas » d'une machine de Turing apparaît alors en fait comme la détermination de l'image par t du couple (e, q), où e désigne l'état courant et q le symbole contenu dans la case sur laquelle pointe la tête de lecture-écriture. C'est d'ailleurs la nature de t qui permet de distinguer entre machine de Turing déterministe et machine de Turing non déterministe.

Machine de Turing déterministe : Une machine de Turing est dite déterministe si l'ensemble des transitions qui lui est associée est constitué de transitions déterministes, c'est-à-dire que la fonction de transitions décrit une application fonctionnelle. Autrement dit, pour tout couple (e, q), où e désigne l'état courant et q le symbole lu dans la case courante, la fonction de transitions t associe un unique triplet (e', q', d) où e' représente le nouvel état d'arrivée, q' le symbole à écrire sur la bande à la place de q et d le déplacement à effectuer.

Machine de Turing non déterministe : La seule différence entre une machine de Turing déterministe et une machine de Turing non déterministe porte sur la fonction de transition. Alors que, pour la première, la fonction de transition t associe un seul triplet (e', q', d) à tout couple (e, q), t peut en associer plusieurs pour une machine de Turing non déterministe. Plus formellement, la fonction de transition d'une telle machine est définie de $\mathcal{E} \times \mathcal{S}$ dans $\mathcal{P}(\mathcal{E} \times \mathcal{S} \times \{G, D\})$. C'est en ce point que réside le caractère aléatoire de la machine de Turing non déterministe : à chaque pas, un triplet définissant l'instruction à exécuter parmi les possibilités associées à (e, s) est choisi aléatoirement.

En dépit de la contradiction apparente des termes, une machine de Turing déterministe est un cas particulier de machine de Turing non déterministe : c'est le cas pour lequel l'ensemble des triplets possibles est un singleton quel que soit le couple (e, s).

Bien que d'apparence très frustres, les machines de Turing fournissent un modèle universel pour les ordinateurs et les algorithmes. En fait, elles permettent de calculer toutes les fonctions récursives [1] et la « thèse de Church » (de ce fait présentée parfois comme la « thèse de Church-Turing ») peut se reformuler, en termes non techniques de la manière suivante : tout ce qui est calculable l'est avec une machine de Turing. Plus précisément, il est possible de montrer que tout calcul nécessitant un temps $f(n)$ sur une machine à accès aléatoire [2] peut être traduit sur une machine de Turing effectuant le même calcul en temps $\mathcal{O}(f(n)^6)$, voire même $\mathcal{O}(f(n)^3)$ avec une machine à plusieurs rubans (Papadimitriou 1994).

Malgré cette augmentation non négligeable du temps nécessaire à l'exécution d'un programme, cette propriété permet l'utilisation de la machine de Turing comme référence dans les définitions théoriques des classes de complexité.

1.1.3 Classes de complexité des problèmes de décision

Nous allons maintenant nous intéresser à l'utilisation que nous pouvons faire des machines de Turing pour définir des classes de problèmes selon leurs difficultés intrinsèques, ce que l'on appelle de nouveau complexité (des problèmes et non plus des algorithmes).

1.1.3.1 Complexité et problème de décision

Il est possible de définir la complexité d'un problème (comme énoncée précédemment) en fonction d'une machine de Turing déterministe. Dans ce cas, la complexité spatiale et la complexité temporelle sont respectivement évaluées en fonction du nombre de cases du ruban et du nombre de transitions nécessaires à la résolution du problème.

Définition 1.5 (TIME)**.** *Un problème appartient à la classe de complexité $TIME(f(n))$ si et seulement s'il existe une machine de Turing déterministe le résolvant en temps $\mathcal{O}(f(n))$.*

Définition 1.6 (SPACE)**.** *Un problème appartient à la classe de complexité $SPACE(f(n))$ si et seulement s'il existe une machine de Turing déterministe le résolvant en espace $\mathcal{O}(f(n))$.*

De manière générale, la théorie de la complexité ne traite que des *problèmes de décision* (parfois aussi appelés *problèmes de reconnaissance*), c'est-à-dire posant une question dont la réponse est « Oui » ou

1. Les fonctions récursives désignent la classe des fonctions calculables, autrement dit les fonctions dont les valeurs peuvent être calculées à partir de leurs paramètres par un processus mécanique.

2. Les ordinateurs actuels sont conçus d'après un modèle de machine à accès aléatoire. Ces machines possédant une mémoire adressable ont un accès direct à l'information, contrairement à la machine de Turing qui est obligée de parcourir séquentiellement son ruban pour obtenir un résultat intermédiaire.

« Non ». En effet, de nombreux problèmes informatiques peuvent se réduire à des problèmes de décision. Un problème dont la réponse n'est ni « Oui » ni « Non » peut être simplement transformé en un problème de décision (typiquement « existe-t-il une solution au problème ? »).

Définition 1.7 (Problème de décision)**.** *Un problème* Π *de décision est un problème n'ayant que deux solutions possibles : « Oui » ou « Non ». C'est-à-dire que l'ensemble* D_Π *des instances de* Π *peut être scindé en deux ensembles disjoints :*

1. Y_Π *: l'ensemble des instances telles qu'il existe un programme les résolvants et répondant « Oui » ;*
2. N_Π *: l'ensemble des instances pour lesquelles la réponse est « Non ».*

Définition 1.8 (Problème de décision complémentaire)**.** *Le problème complémentaire d'un problème de décision* Π *est le problème de décision* Π^c *tel que :*

$$D_{\Pi^c} = D_\Pi \text{ et } Y_{\Pi^c} = N_\Pi.$$

Dans ce qui suit nous présentons les différentes classes de complexité des problèmes de décision.

1.1.3.2 Classes de complexité

La théorie de la complexité repose sur la définition de classes de complexité qui permettent d'ordonner les problèmes en fonction des algorithmes qui existent pour les résoudre.

Définition 1.9 (Classe P)**.** *La classe de complexité* P *regroupe l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par une machine de Turing déterministe.*

Par définition, la classe P (pour polynomial) contient les problèmes de décision polynomiaux, c'est-à-dire que l'on peut résoudre à l'aide d'un algorithme déterministe en temps polynomial par rapport à la taille de l'instance. S'il est vrai que la classe P contient de nombreux problèmes de décision, certains autres ne sont pas connus pour être polynomiaux (ce qui ne signifie pas qu'ils ne le sont pas). Nous allons maintenant définir une classe plus vaste : la classe NP.

Définition 1.10 (Classe NP)**.** *La classe de complexité* NP *regroupe l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par une machine de Turing non déterministe.*

Les problèmes de la classe NP (pour *Non déterministe Polynomial*) sont les problèmes de décision pour lesquels la réponse « Oui » peut être décidée par un algorithme non déterministe en un temps polynomial par rapport à la taille de l'instance. De façon équivalente, c'est la classe des problèmes qui admettent un algorithme polynomial qui, étant donnée une solution du problème NP (certificat), sont capables de répondre si « Oui » ou « Non » cette solution est une réponse au problème. Il est possible de définir le problème complémentaire (dual) de tout problème NP.

Définition 1.11 (Classe $CoNP$)**.** *La classe* $CoNP$ *regroupe l'ensemble des problèmes de décision dont les problèmes complémentaires appartiennent à la classe* NP*.*

Puisqu'une machine de Turing déterministe peut être considérée comme une machine de Turing non déterministe particulière nous obtenons immédiatement la propriété suivante :

Propriété 1.1. *Nous avons :* $P \subseteq NP$ *et* $P \subseteq CoNP$*.*

La question de savoir si ces inclusions sont strictes ou s'il s'agit d'une égalité constitue l'une des questions ouvertes fondamentales de la théorie de la complexité. Sa résolution aurait bien sûr des conséquences en théorie de la calculabilité, mais aussi des répercussions pratiques dans tous les domaines où interviennent l'informatique et l'ordinateur, c'est-à-dire finalement à peu près tous les domaines ...

Dans la partie suivante, nous présentons la notion de complétude. Cette notion permet de définir l'ensemble des problèmes les plus difficiles d'une classe.

1.1.3.3 Réduction et complétude

Dans le cadre des problèmes décidables, une réduction permet de ramener la décidabilité d'un problème à celle d'un autre aussi difficile. Cette notion de difficulté s'énonce de la manière suivante :

Définition 1.12 (Problème C-difficile)**.** *Soit C une classe de complexité, un problème est dit C-difficile si et seulement s'il est au moins aussi difficile que tous les problèmes dans C.*

Exemple 1.3. *Décider de l'appartenance d'un élément dans un tableau est un problème P-difficile.*

Remarque 1.3. *Tout problème dans NP est P-difficile.*

Comme nous avons pu le remarquer précédemment, ce n'est pas parce qu'un problème est dans une classe C qu'il est difficile pour cette classe (par exemple tout problème Π de P est dans NP mais ce n'est pas pour autant que Π est difficile pour la classe NP). Afin de définir une véritable classification des problèmes une notion de réduction fonctionnelle polynomiale est utilisée. Intuitivement, la notion de réduction permet d'établir qu'un problème est aussi « dur » qu'un autre si un premier problème se réduit « facilement » en un second alors le premier problème est (au moins) aussi dur que le second.

Définition 1.13 (Réduction fonctionnelle polynomiale)**.** *Soient deux problèmes Π et Π', et un algorithme f qui prend en entrée des instances de Π et qui retourne des instances de Π'. f est appelée réduction fonctionnelle polynomiale si et seulement si :*

- *f est un algorithme polynomial ;*
- *π est une instance positive de Π si et seulement si $f(\pi)$ est une instance positive de Π'.*

Définition 1.14 (Complétude)**.** *Un problème Π est complet pour sa classe de complexité C si et seulement si pour tout problème $\Pi' \in C$, il existe une réduction fonctionnelle polynomiale de Π' vers Π. Un tel problème est dit C-Complet.*

Il est possible de situer schématiquement (voir figure 1.2) les classes de complexité P, NP et $CoNP$ ainsi que leur complétude en se basant sur les deux conjectures suivantes (Papadimitriou 1994) :

- Avons nous $P = NP$?
- Avons nous $CoNP = NP$?

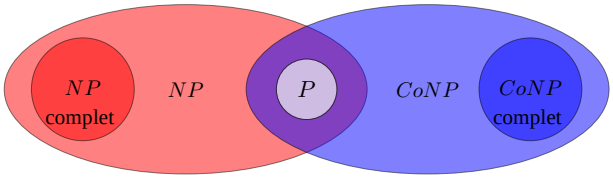


FIGURE 1.2 – Structuration possible des classes P, NP et $CoNP$ ainsi que leur complétude.

Une investigation plus profonde nous conduit à nous interroger sur la possibilité de poursuivre la classification des problème de décision au delà de NP. Cette classification, appelée hiérarchie polynomiale, a pour base les classes P, $CoNP$ et NP et peut être définie à l'aide d'une machine de Turing à Oracle. Comme son nom l'indique, une machine de Turing à oracle a pour mission de deviner une solution permettant de justifier la réponse « oui ». Formellement, une telle machine se définit comme suit :

Définition 1.15 (Machine de Turing à oracle)**.** *Une machine de Turing à oracle est une machine de Turing (déterministe ou non) à laquelle 3 étapes sont ajoutées : $q_?$ qui sert à interroger l'oracle, q_{oui} et q_{non} qui représentent la réponse de l'oracle. Un langage oracle $\mathcal{A}$ est également définie et un ruban de la machine, appelée bande de l'oracle, lui est réservé. Cet oracle permet de décider si un élément x appartient à $\mathcal{A}$ en une étape de calcul.*

Notation 1.1 (Classe C^A)**.** *Soit C une classe de complexité quelconque déterministe ou non, C^A est la classe des langages décidés dans un temps borné comme dans C, mais avec l'appel à un oracle A.*

Le langage A peut être une classe de complexité. Ainsi P^{NP} représente l'ensemble de langages décidés en temps polynomial par une machine de Turing déterministe en faisant appel à un oracle capable de répondre en un pas de calcul aux questions pour tous les langages décidés en temps polynomial par une machine de Turing non déterministe. À l'aide de cette notion il est possible de définir un nouvel ensemble de classes de manière récursive :

Définition 1.16 (Hiérarchie polynomiale)**.** *La hiérarchie polynomiale (Stockmeyer 1976), est une classe de complexité définie récursivement à partir de ses sous-classes à l'aide des machines de Turing avec oracle de la manière suivante :*

- $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$;
- $\Delta_{k+1}^P = P^{\Sigma_k^P}$;
- $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$;
- $\Pi_{k+1}^P = Co\Sigma_{k+1}^P$.

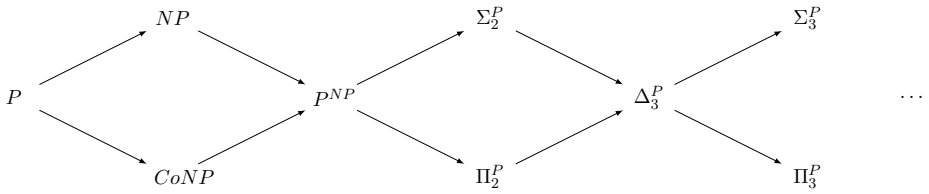


FIGURE 1.3 – Représentation graphique de la hiérarchie polynomiale. Les flèches indiquent l'inclusion d'une classe de complexité dans un autre.

Comme nous pouvons le voir sur la figure 1.3, les problèmes sont classés de façon incrémentale, la classe d'un nouveau problème étant déduite d'un ancien problème. Il a toutefois été nécessaire de définir un « premier » problème NP-complet afin de classer tous les autres. Ce premier problème à avoir été démontré NP-complet, par Cook (1971), est le problème de la satisfaisabilité propositionnelles (SAT).

1.2 Le problème SAT

Le problème de satisfaisabilité d'une formule logique booléenne ou « problème SAT » est un problème de décision au formalisme simple basé sur la logique propositionnelle. Le langage de la logique

propositionnelle, introduit par Boole (1854) il y a un peu plus de 150 ans, est une manière naturelle de représenter des connaissances. Construit à partir d'un alphabet et de règles de syntaxe, il peut être utilisé dans un système syntaxique sans se soucier du sens des symboles manipulés, il s'agit alors de *théorie de la preuve*, ou par l'intermédiaire d'une sémantique, il s'agit de *théorie des modèles*.

La résolution du problème SAT est fondamentale à la fois théoriquement et pratiquement. Théoriquement car il occupe un rôle prépondérant en théorie de la complexité, où il représente le problème NP-complet de référence (Cook 1971). Pratiquement, car de nombreux problèmes s'y ramènent naturellement ou le contiennent (exemple : *planification classique*, *model checking*, *bioinformatique*, etc.).

Dans la suite, nous décrivons le langage de la logique propositionnelle et sa sémantique de manière à définir le problème SAT. Une fois ce dernier défini, nous présentons un problème connexe : la détection de noyaux minimalement inconsistants.

1.2.1 Syntaxe : le langage propositionnel

Nous commençons par définir les principaux éléments syntaxiques de la logique propositionnelle. Pour spécifier la syntaxe, il est nécessaire de définir les symboles pouvant être utilisés pour formuler des énoncés. Ces symboles forment alors l'alphabet du langage :

Définition 1.17 (Alphabet de la logique propositionnelle)**.** *L'alphabet de la logique propositionnelle est constitué :*

- *des symboles* $\bot$ *et* $\top$ *représentant respectivement les constantes propositionnelles « faux » et « vrai » ;*
- *d'un ensemble infini dénombrable de symboles propositionnels, noté* PS *;*
- *de l'ensemble des connecteurs logiques usuels : le symbole* $\wedge$ *est utilisé pour la conjonction,* $\vee$ *pour la disjonction,* $\neg$ *pour la négation,* $\Rightarrow$ *pour l'implication,* $\Leftrightarrow$ *pour la bi-implication et* $\oplus$ *pour le ou exclusif ;*
- *des symboles de ponctuation "(" et ")".*

L'ensemble des formules finies formées à partir de cet alphabet est noté PROP_{PS} et est défini inductivement de la manière suivante :

Définition 1.18 (Formule propositionnelle)**.** *L'ensemble des formules bien formées de* PROP_{PS} *est le plus petit ensemble tel que :*

- *tout élément de* $\text{PS} \cup \{\bot, \top\}$ *est élément de* PROP_{PS} *;*
- *si* $\alpha \in \text{PROP}_{\text{PS}}$ *alors* $(\neg\alpha)$ *est élément de* PROP_{PS} *;*
- *si* α *et* β *sont des éléments de* PROP_{PS} *alors* $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \Rightarrow \beta)$, $(\alpha \Leftrightarrow \beta)$ *et* $(\alpha \oplus \beta)$ *sont éléments de* PROP_{PS}.

Exemple 1.4. *Soit* $\{a, b, c, d\}$ *un ensemble de symboles propositionnels, alors* $(((a \Leftrightarrow b) \vee (\neg d)) \wedge c)$ *appartient à* PROP_{PS}*, contrairement à* $(((a \Leftrightarrow b)$ *et* $((\neg d)\wedge)$.

Par souci de clarté, il est courant d'omettre certaines parenthèses à condition de ne pas rendre la proposition ambiguë. Il arrive aussi que certaines parenthèses soient remplacées par les symboles "[" et "]" afin de rendre plus visibles certaines formules ou sous-formules. Sur notre exemple la première formule peut s'écrire : $[((a \Leftrightarrow b) \vee (\neg d)) \wedge c]$. Nous choisissons dans la suite de ce manuscrit d'adopter les conventions suivantes :

- les variables sont représentées par des lettres latines minuscules ;
- les ensemble de variables sont représentés par des lettres latines majuscules ;

- les formules sont représentées par des lettres grecques minuscules ;
- les ensembles de formules sont représentés par des lettres grecques majuscules ;
- soit Σ une formule propositionnelle, $\mathcal{V}_\Sigma$ dénote l'ensemble des variables propositionnelles de Σ ;
- la négation d'une formule Σ est noté $\overline{\Sigma}$.

Nous définissons à présent les notions de formule indépendante, de littéral et de littéral pur.

Définition 1.19 (Formules indépendantes)**.** *Les formules propositionnelles $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ sont dites indépendantes si et seulement si $\forall i, j$ tel que $1 \leq i < j \leq n$, $\mathcal{V}_{\Sigma_i} \cap \mathcal{V}_{\Sigma_j} = \emptyset$.*

Définition 1.20 (Littéral)**.** *Un* littéral *est un symbole propositionnel ou bien sa négation. Soit ℓ un symbole propositionnel, alors ℓ est appelé* littéral positif*, $\neg\ell$ est appelé* littéral négatif*, ℓ et $\neg\ell$ sont des* littéraux complémentaires *et le complémentaire de ℓ est noté $\tilde{\ell}$.*

Définition 1.21 (littéral pur ou monotone)**.** *Un littéral ℓ est dit* pur *(ou* monotone*) pour une formule propositionnelle Σ si et seulement si ℓ apparaît dans Σ et $\tilde{\ell}$ n'apparaît pas dans Σ.*

Dans la section suivante nous nous intéressons à la sémantique des formules bien formées, c'est-à-dire sous quelles conditions un énoncé est « vrai » ou « faux ».

1.2.2 Sémantique

La sémantique de la logique propositionnelle associe une signification à ses formules et explique les conditions qui rendent les formules vraies ou fausses. La notion de vérité d'une formule est régie par deux postulats. Premièrement, le postulat de bivalence permet d'établir que la valeur de vérité d'un symbole propositionnel appartient à l'ensemble des valeurs de vérité $\mathbb{B} = \{vrai, faux\}$. Deuxièmement, le postulat de vérifonctionnalité permet de déterminer la valeur de vérité d'une formule complexe en fonction des valeurs de vérité de ses composants immédiats. Il s'ensuit que pour déterminer la valeur de vérité d'une formule il suffit de connaître les valeurs de vérité de ces composantes les plus simples, c'est-à-dire les constantes et les opérateurs. Nous définissons $\bot = faux$, $\top = vrai$ et la sémantique de l'ensemble des opérateurs logiques comme reporté dans le tableau 1.2.

x	y	$\neg x$	$(x \vee y)$	$(x \wedge y)$	$(x \Rightarrow y)$	$(x \Leftrightarrow y)$	$(x \oplus y)$
$vrai$	$vrai$	$faux$	$vrai$	$vrai$	$vrai$	$vrai$	$faux$
$vrai$	$faux$	$faux$	$vrai$	$faux$	$faux$	$faux$	$vrai$
$faux$	$vrai$	$vrai$	$vrai$	$faux$	$vrai$	$faux$	$vrai$
$faux$	$faux$	$vrai$	$faux$	$faux$	$vrai$	$vrai$	$faux$

TABLE 1.2 – Sémantique usuelle des opérateurs logique.

La sémantique permet de définir des règles d'interprétations pour les formules à partir de la valeur de vérité de chacune des propositions qui les composent. Ces règles sont les suivantes :

Définition 1.22 (Interprétation)**.** *Une interprétation $\mathcal{I}$ est une application de* PS *dans $\mathbb{B}$ qui attribue une valeur de vérité à chaque symbole propositionnel :*

$$\mathcal{I} : \text{PS} \rightarrow \mathbb{B}$$

Soit $\mathcal{I}$ une interprétation, la sémantique des propositions selon cette interprétation $\mathcal{I}$ est définie inductivement telle que $\forall \alpha, \beta \in \text{PROP}_{\text{PS}}$:

- *$\mathcal{I}(\top) = vrai$ et $\mathcal{I}(\bot) = faux$;*

- $\mathcal{I}(\neg\alpha) = vrai$ *si et seulement si* $\mathcal{I}(\alpha) = faux$ *;*
- $\mathcal{I}(\alpha \circ \beta) = \mathcal{I}(\alpha) \circ \mathcal{I}(\beta)$ *tel que* $\circ \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus\}$ *respecte la sémantique énoncée précédemment.*

Comme nous pouvons le voir, l'interprétation d'une formule est définie par l'interprétation des variables qui la composent. Cette interprétation est communément représentée comme l'ensemble des littéraux interprétés à vrai et dans ce cas la sémantique des opérateurs ensemblistes est conservée.

Exemple 1.5. *Soient* $\Sigma = \{(a \vee b) \wedge (a \Rightarrow c)\}$ *une formule propositionnelle et* $\mathcal{I} = \{a, b, \neg c\}$ *une interprétation de* Σ*. Nous avons* $\mathcal{I}(\Sigma) = \mathcal{I}(\top \vee \top) \wedge \mathcal{I}(\top \Rightarrow \bot) = \top \wedge \bot = \bot$.

L'ensemble des interprétations, noté $\mathfrak{S}$, pouvant être construit sur les variables d'une formule propositionnelle Σ a un cardinal de $2^{\mathcal{V}_\Sigma}$. Nous définissons la distance entre deux interprétations $\mathcal{I}$ et $\mathcal{I}'$ de $\mathfrak{S}$ comme suit :

Définition 1.23 (Distance de Hamming)**.** *Soient* $\mathcal{I}$ *et* $\mathcal{I}'$ *deux interprétations d'une formule propositionnelle* Σ*. La distance de Hamming entre* $\mathcal{I}$ *et* $\mathcal{I}'$*, notée* $\delta_\mathfrak{h}(\mathcal{I}, \mathcal{I}')$*, est définie par* $|\zeta_\mathfrak{h}(\mathcal{I}, \mathcal{I}')|$ *tel que* $\zeta_\mathfrak{h}(\mathcal{I}, \mathcal{I}') = \{x \in \mathcal{V}_\Sigma$ *telle que* $\mathcal{I}(x) \neq \mathcal{I}'(x)\}$.

Exemple 1.6. *Soient* $\Sigma = a \vee (b \wedge c)$*,* $\mathcal{I} = \{a, b, c\}$ *et* $\mathcal{I}' = \{\neg a, \neg b, c\}$ *;* $\zeta_\mathfrak{h}(\mathcal{I}, \mathcal{I}') = \{a, b\}$ *et* $\delta_\mathfrak{h}(I, \mathcal{I}') = 2$.

Une interprétation n'est pas forcément définie sur l'ensemble des variables propositionnelles de la formule. Dans ce cas, l'interprétation peut être partielle ou incomplète, ce qui se définit formellement de la manière suivante :

Définition 1.24 (interprétation partielle, complète et incomplète)**.** *Soit* Σ *une formule propositionnelle, une interprétation* $\mathcal{I}$ *construite sur les variables de* Σ *est dite :*

- *partielle si* $|\mathcal{I}| \leq |\mathcal{V}_\Sigma|$ *;*
- *complète si* $|\mathcal{I}| = |\mathcal{V}_\Sigma|$ *;*
- *incomplète si* $|\mathcal{I}| < |\mathcal{V}_\Sigma|$.

Remarque 1.4. *Dans la suite de ce manuscrit, lorsqu'aucune information est apportée sur la nature de l'interprétation, elle est considérée comme complète.*

Il est aussi possible de prolonger une interprétation partielle vers une interprétation complète de la manière suivante :

Définition 1.25 (prolongement d'une interprétation partielle)**.** *Soient* Σ *une formule propositionnelle ainsi que* $\mathcal{I}_p$ *et* $\mathcal{I}$ *respectivement une interprétation partielle et complète de* Σ*. Le prolongement de* $\mathcal{I}_p$ *par* $\mathcal{I}$ *est l'interprétation complète* $\mathcal{I}'_p$ *telle que* $\mathcal{I}_p \cup \mathcal{I}'_p = \mathcal{I}$ *et* $\forall \ell \in \mathcal{I}'_p$*,* $\{\ell, \neg\ell\} \cap \mathcal{I}_p = \emptyset$.

La définition suivante permet d'établir les notions de modèle, de satisfaisabilité, d'impliquant, d'impliqué et de conséquence logique d'une formule propositionnelle.

Définition 1.26 (Modèle, satisfaisabilité, impliquant, impliqué et conséquence logique)**.** *Soient* Σ *et* Ψ *deux formules, nous avons :*

- *un* ***modèle*** $\mathcal{I}$ *de* Σ*, noté* $\mathcal{I} \models \Sigma$*, est une interprétation telle que l'application de la sémantique vérifie* Σ*, c'est-à-dire* $\mathcal{I}(\Sigma) = \top$ *;*
- *une interprétation* $\mathcal{I}$ ***falsifie*** Σ*, notée* $\mathcal{I} \not\models \Sigma$*, si l'application de la sémantique ne vérifie pas* Σ*, c'est-à-dire* $\mathcal{I}(\Sigma) = \bot$ *;*

- Σ *est* ***satisfiable*** *si elle admet au moins un modèle, sinon elle est* ***insatisfiable*** *;*
- *un* ***impliquant*** ρ *de* Σ *est une conjonction de littéraux telle que l'interprétation* $\mathcal{I} = \{\ell \in \rho\}$ *construite sur les littéraux de* ρ *est modèle de* Σ*. Un* ***impliquant premier*** *de* Σ *est un impliquant qui est minimal pour l'inclusion ;*
- *si tout modèle de* Ψ *est modèle de* Σ*, noté* $\Psi \models \Sigma$*, alors* Σ *est une conséquence logique de* Ψ *;*
- *un littéral* $\ell \in \mathcal{V}_\Sigma$ *est impliqué par* Σ *si et seulement si* $\Sigma \models \ell$*. Dans ce cas le littéral* ℓ *est vrai dans tout les modèles de* Σ *;*
- *un* ***impliqué*** α *de* Σ *est une disjonction de littéraux telle que* $\Sigma \models \alpha$*. Un* ***impliqué premier*** *de* Σ *est un impliqué qui est minimal pour l'inclusion ;*
- Ψ *et* Σ *sont logiquement équivalentes, noté* $\Psi \equiv \Sigma$*, si* $\Psi \models \Sigma$ *et* $\Sigma \models \Psi$*.*

Le théorème suivant permet d'énoncer un résultat fondamental en démonstration automatique (preuve par l'absurde).

Théorème 1.2 (Déduction)**.** *Soient* Σ *et* Ψ *deux formules propositionnelles,* $\Sigma \models \Psi$ *si et seulement si* $\Sigma \wedge \neg\Psi$ *est une formule insatisfiable.*

Ce théorème montre que prouver l'implication sémantique revient à prouver l'inconsistance d'une formule. Il est très important en pratique puisque la grande majorité des algorithmes de démonstration automatique présentés dans le chapitre 3 exploitent ce théorème.

Dans la section suivante nous présentons les différentes formes normales et nous énonçons le fait que toute formule propositionnelle peut être définie sous l'une de ces formes.

1.2.3 Formes normales

Dans cette section nous nous intéressons aux différentes formes normales. Plus particulièrement nous énonçons un théorème permettant de limiter l'établissement de la satisfaction d'une formule propositionnelle au cas d'une formule mise sous forme normale conjonctive.

Nous commençons par introduire les notions de clauses et de termes :

Définition 1.27 (clause et terme)**.** *Soit* PS *un ensemble de variables, nous avons :*

- *une* ***clause*** *est une disjonction de littéraux, c'est-à-dire une formule de la forme* $(\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n)$ *où chaque* ℓ_i *est un littéral associé à une variable de* PS *;*
- *un* ***terme*** *est une conjonction de littéraux, c'est-à-dire une formule de la forme* $(\ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n)$ *où chaque* ℓ_i *est un littéral associé à une variable de* PS*.*

Remarque 1.5. *Lorsqu'il n'y a aucune ambiguïté, les clauses et les termes sont représentés comme des ensembles de littéraux. Dans ce cas, comme pour les interprétations, la signification des symboles ensemblistes définie sur les littéraux par rapport aux clauses et aux termes est conservée.*

Nous définissons à présent trois cas particuliers de formules propositionnelles.

Définition 1.28 (Formes Normales)**.** *Nous distinguons trois formes normales particulières pour les propositions :*

- *une proposition est sous forme normale négative (*NNF *pour « Negative Normal Form ») si elle est exclusivement constituée de conjonctions, de disjonctions et de littéraux ;*
- *une proposition est sous forme normale disjonctive (*DNF *pour « Disjonctive Normal Form ») si c'est une disjonction de termes ;*

– *une proposition est sous forme normale conjonctive (*CNF *pour « Conjonctive Normal Form ») si c'est une conjonction de clauses.*

Exemple 1.7. *Les formules* $[(a \vee b) \vee ((\neg c) \wedge d)]$, $[(a \wedge b) \vee ((\neg c) \wedge d)]$ *et* $[(a \vee b) \wedge ((\neg c) \vee d)]$ *sont respectivement sous forme normale négative, disjonctive et conjonctive.*

Remarque 1.6. *De la même manière que les clauses et les termes, les formules* CNF *et* DNF *sont respectivement représentées comme des ensembles de clauses et de termes.*

Comme nous pouvons le remarquer les formes normales conjonctives et disjonctives sont des cas spécifiques de la forme normale négative. De plus, la propriété suivante spécifie que toute formule peut être mise sous forme normale.

Propriété 1.3. *Toute formule de la logique propositionnelle peut être réécrite sous une forme normale.*

Cependant la transformation classique peut nécessiter une croissance exponentielle de la taille de l'ensemble obtenu. Néanmoins, Tseitin (1968) propose une approche permettant de transformer en temps et espace linéaire toute formule en une formule sous forme normale conjonctive équivalente du point de vue de la satisfaisabilité. Cette transformation permet de focaliser les recherches sur le problème SAT au cas de formule CNF.

Définition 1.29 (Le problème SAT)**.** *Le problème* SAT *est le problème de décision qui consiste à savoir si une formule sous forme normale conjonctive possède ou pas un modèle.*

Dans la suite de ce manuscrit, lorsqu'il est fait référence à une formule propositionnelle, sauf mention contraire, il est admis qu'il s'agit d'une formule sous forme normale conjonctive.

À présent, nous soulignons quelques particularités concernant l'aspect syntaxique et sémantique associé aux clauses.

Définition 1.30 (Aspect syntaxique des clauses)**.** *Soit* α *une clause, nous avons que* α *est :*

– ***satisfaite*** *par une interprétation* $\mathcal{I}$ *si au moins un littéral de* α *est affecté à vrai. Nous notons* $\mathcal{L}^+_{\mathcal{I}}(\alpha)$ *(respectivement* $\mathcal{L}^-_{\mathcal{I}}(\alpha)$*) l'ensemble des littéraux positifs (respectivement négatifs) de* α *pour l'interprétation* $\mathcal{I}$*, c'est-à-dire* $\mathcal{L}^+_{\mathcal{I}}(\alpha) = \mathcal{I} \cap \alpha$ *(respectivement* $\mathcal{L}^-_{\mathcal{I}}(\alpha) = \mathcal{I} \cap (\overline{\alpha})$*) ;*
– ***falsifiée*** *par une interprétation* $\mathcal{I}$ *si tous les littéraux de* α *sont affectés à faux ;*
– ***unisatisfaite*** *par une interprétation* $\mathcal{I}$ *si un seul littéral de* α *est affecté à vrai et que les autres sont affectés à faux ;*
– ***unitaire***, ***binaire***, ***ternaire*** *et* ***n-aire*** *si et seulement si elle contient respectivement exactement un, deux, trois et* $n > 3$ *littéraux différents ;*
– ***positive***, ***négative*** *ou* ***mixte*** *si et seulement si elle est constituée respectivement uniquement de littéraux positifs, négatifs ou positifs et négatifs ;*
– ***vide***, *noté* $\perp$, *si et seulement si elle ne contient aucun littéral. Elle est par définition insatisfiable ;*
– ***tautologique*** *si et seulement si elle contient deux littéraux complémentaires. Elle est par définition satisfaite ;*
– ***Horn*** *(respectivement* ***reverse-Horn****) si et seulement si elle contient au plus un littéral positif (respectivement négatif).*

Dans certaines conditions, le choix des clauses utilisées dans une formule peuvent influencer sa complexité. La définition suivante permet d'établir certaines restrictions de problème SAT.

Définition 1.31 (Différents types d'instances)**.** *Soit* Σ *une instance* SAT*,* Σ *est une instance :*

- *k-SAT si toutes les clauses contiennent exactement $k \in \mathbb{N}$ littéraux. Hormis* 1-SAT *et* 2-SAT *(Cook 1971), le problème k-SAT est généralement NP-complet ;*
- *Horn-SAT (respectivement Reverse-Horn-SAT) si toutes les clauses de Σ sont Horn (respectivement reverse-horn). En ce qui concerne le test de satisfaisabilité d'un ensemble de clauses de Horn ou reverse-horn, plusieurs auteurs ont montré que la résolution est polynomiale voir même linéaire (Minoux 1988, Dalal 1992, Rauzy 1995) ;*
- *Horn-renommable si et seulement s'il existe un renommage ρ des littéraux de Σ tel que $\rho(\Sigma)$ soit une formule Horn. Déterminer un tel renommage [3] est un problème polynomial (Lewis 1978).*

Plusieurs techniques peuvent être appliquées sur les clauses d'une formule Σ sans pour autant en changer sa satisfaisabilité. Ces techniques permettent soit de supprimer des clauses soit de supprimer des littéraux de clauses de Σ.

La première approche présentée est la méthode de résolution. Informellement, considérons deux clauses $(\alpha \vee x)$ et $(\beta \vee \neg x)$. En supposant qu'il existe une interprétation qui satisfait ces clauses, si cette interprétation satisfait x (respectivement $\neg x$) alors nous pouvons en déduire qu'elle satisfait β (respectivement α), dans les deux cas cette interprétation satisfait $\alpha \vee \beta$, ce qui s'écrit plus formellement :

Définition 1.32 (Résolution)**.** *Soient $\alpha_1 = (x \vee \beta_1)$ et $\alpha_2 = (\neg x \vee \beta_2)$ deux clauses ayant en commun une variable x présente dans les deux clauses sous la forme de littéraux complémentaires, la nouvelle clause $\alpha = (\beta_1 \vee \beta_2)$ peut être obtenue par la suppression de toutes les occurrences du littéral x et $\neg x$ dans α_1 et α_2. Cette opération, notée $\eta[x, \alpha_1, \alpha_2]$, est appelée résolution et la clause produite est appelée résolvante.*

Exemple 1.8. *Soient $\alpha = (a \vee b \vee c)$ et $\beta = (\neg a \vee c \vee d)$ deux clauses, nous avons $\eta[a, \alpha, \beta] = (b \vee c \vee d)$.*

L'opération de résolution peut être étendue à un ensemble de clauses (Eén et Biere 2005) de la manière suivante.

Définition 1.33 (Résolution appliquée à un ensemble de clauses)**.** *Soient Σ_x un ensemble de clauses contenant le littéral x et $\Sigma_{\neg x}$ un ensemble de clauses contenant le littéral $\neg x$, la résolution de Σ_x et $\Sigma_{\neg x}$ est définie comme suit :*

$$\eta[x, \Sigma_x, \Sigma_{\neg x}] = \{\eta[x, \alpha, \beta] \text{ telle que } \alpha \in \Sigma_x \text{ et } \alpha \in \Sigma_{\neg x}\}$$

Exemple 1.9. *Soient $\Sigma_1 = \{(a \vee b), (a \vee c)\}$ et $\Sigma_2 = \{(\neg a \vee d)\}$ deux formules, nous avons que $\eta[a, \Sigma_1, \Sigma_2] = \{(b \vee d), (c \vee d)\}$.*

La deuxième technique, appelée *subsumption*, permet la suppression de clauses de la formule.

Définition 1.34 (Sous-sommation ou subsumption)**.** *Une clause $\alpha_1 = (a_1 \vee a_2 \vee \cdots \vee a_n)$ subsume une clause $\alpha_2 = (a_1 \vee a_2 \vee \cdots \vee a_n \vee b_1 \vee b_2 \vee \cdots \vee b_m)$ si α_2 contient tous les littéraux de α_1.*

Exemple 1.10. *Soient $\alpha_1 = (a \vee b)$ et $\alpha_2 = (a \vee b \vee c)$ deux clauses, la clause α_1 subsume la clause α_2.*

Il est possible d'effectuer la clôture par subsumption d'une formule. Cette clôture s'énonce de la manière suivante :

Définition 1.35 (Clôture par subsumption)**.** *Une formule Σ est close par subsumption si et seulement si $\forall \alpha_1 \in \Sigma, \nexists \alpha_2 \in \Sigma$ telle que α_2 subsume α_1.*

3. Un renommage consite à renommer un littéral en son complémentaire.

Exemple 1.11. *La formule* $\Sigma = \{(a \vee b), (a \vee c), (\neg a \vee d)\}$ *est close par subsumption.*

La troisième technique présentée est une combinaison des deux précédentes. Cette approche appelée *self-subsumption* est définie formellement de la manière suivante.

Définition 1.36 (Self-subsumption)**.** *La clause* α_1 *self-subsume la clause* α_2 *si et seulement si la résolvante de* α_1 *et* α_2 *subsume* α_2*.*

Exemple 1.12. *Soient* $\alpha_1 = (a \vee b)$ *et* $\alpha_2 = (\neg a \vee b \vee c)$ *deux clauses, les clauses* α_1 *et* α_2 *se résolvent en* a *et produisent la résolvante* $(b \vee c)$ *qui subsume* α_2*.* α_2 *est donc self-subsumée par* α_1 *en* a*.*

Les deux dernières techniques présentées permettent la suppression de clauses dites *redondantes*. Elles sont, dans ces deux cas, syntaxiques mais peut être étendues sémantiquement de la manière suivante :

Définition 1.37 (Clause redondante)**.** *Soient* Σ *une formule* CNF *et* α *une clause de* Σ*,* α *est redondante dans* Σ *si et seulement si* $\Sigma \setminus \{\alpha\} \models \alpha$*.*

Lorsque une base de connaissance ne possède aucune clause redondante il est dit que cette base est irredondante minimale, ce qui s'énonce comme suit :

Définition 1.38 (CNF irredondante minimale)**.** *Soit* Σ *une formule* CNF*,* Σ *est irredondante minimale si et seulement si* $\forall \alpha \in \Sigma$ *alors* $\Sigma \setminus \{\alpha\} \not\models \alpha$*.*

La dernière méthode présentée, consiste à simplifier une formule par un littéral. Cette approche est différente des dernières puisqu'elle peut changer la satisfaisabilité de la formule considérée.

Définition 1.39 (Formule simplifiée par un littéral)**.** *Soit* Σ *une formule, nous notons* $\Sigma_{|\ell}$ *la formule obtenue en éliminant de* Σ *les clauses où le littéral* ℓ *apparaît et en supprimant l'occurrence du littéral* $\bar{\ell}$ *des clauses le contenant. Formellement nous avons :*

$$\Sigma_{|\ell} = \{\alpha | \alpha \in \Sigma \text{ et } \{\ell, \neg\ell\} \cap \alpha = \emptyset\} \cup \{\alpha \setminus \{\ell\} | \alpha \in \Sigma, \neg\ell \in \alpha\}$$

Exemple 1.13. *Soit* $\Sigma = \{(a \vee b \vee \neg c), (\neg a \vee c \vee \neg d), (\neg a \vee \neg b \vee d)\}$*,* $\Sigma_{|\neg a} = (b \vee \neg c)$*.*

Il est possible d'étendre cette notion à un ensemble de littéraux de la manière suivante :

Définition 1.40 (Formule simplifiée par un ensemble de littéraux)**.** *Soit* Σ *une formule propositionnelle et* $\mathcal{A} = \{x_1, x_2, \ldots, x_n\}$ *un ensemble de littéraux,* $\Sigma_{|\mathcal{A}} = (\ldots((\Sigma_{|x_1})_{|x_2})\ldots)_{|x_n}$ *est la formule obtenue par l'application successive de la simplification de* Σ *par l'ensemble des litéraux de* $\mathcal{A}$*.*

Remarque 1.7. *Cette définition s'étend naturellement aux interprétations.*

Exemple 1.14. *Soient* Σ *la* CNF *de l'exemple 1.13 et* $\mathcal{I} = \{a, b\}$*, nous obtenons* $\Sigma_{|\mathcal{I}} = \{(c \vee \neg d), (d)\}$*.*

Comme nous l'avons souligné, le problème SAT consiste à déterminer si une formule possède ou non une solution. Dans le cas où une solution existe, la plupart des méthodes fournissent un modèle. En revanche, lorsque la formule est insatisfiable, aucun certificat permettant d'expliquer les raisons de ce résultat n'est fourni. Or, il est possible que seules quelques clauses soient réellement responsables de la contradiction. C'est la détection de cet ensemble de clauses, appelé *noyau inconsistant*, qui est discutée dans la section suivante.

1.2.4 Noyaux minimalement inconsistants (MUS)

Extraire d'une formule insatisfiable les raisons de sa non satisfaisabilité peut se révéler utile dans de nombreux domaines (Hunter et Konieczny 2008, Benferhat *et al.* 2005, Besnard *et al.* 2010). Une telle information, de nature explicative, peut être fournie par la localisation de sous-formules minimalement insatisfiables (ou MUS pour « *Minimally Unsatisfiable Subformulae*). Un tel ensemble est défini formellement de la manière suivante :

Définition 1.41 (Formule Minimalement Insatisfiable)**.** *Soit Σ une formule* CNF*, l'ensemble de clauses Γ est un* MUS *de Σ et seulement si il satisfait les conditions suivantes :*

1. $\Gamma \subseteq \Sigma$ *;*
2. Γ *est insatisfiable ;*
3. $\forall \Psi \subset \Gamma$, Ψ *est satisfiable.*

Comme nous l'avons souligné précédemment, la découverte d'un MUS au sein d'une instance insatisfiable peut s'avérer d'une grande utilité pour de nombreuses applications. Malheureusement, ce problème est algorithmiquement difficile. En effet, le simple fait de tester si une formule CNF est un MUS est un problème Δ_2^P-complet (Papadimitriou et Wolfe 1988) [4].

Exemple 1.15. *Soit $\Sigma = \{(\neg d \vee e), (b \vee \neg c), (\neg d), (\neg a \vee b), (a), (a \vee \neg c \vee \neg e), (\neg a \vee c \vee d), (\neg b)\}$ une formule* CNF *insatisfiable composée de 8 clauses construites sur 5 variables. La figure 1.4 schématise, sous forme d'un diagramme de Venn, la formule et ces deux* MUS.

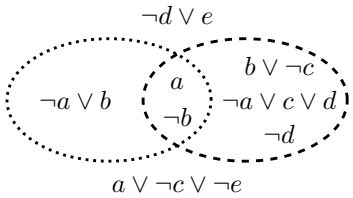


FIGURE 1.4 – Ensemble des MUS d'une formule insatisfiable.

Comme le montre la figure 1.4, une formule CNF peut posséder plusieurs MUS. En fait, elle peut même en contenir un nombre exponentiel par rapport à la taille de la formule. En effet, une CNF de n clauses peut contenir jusqu'à $C_n^{\frac{n}{2}}$ MUS dans le pire cas. Ce nombre potentiellement élevé de MUS au sein d'une même formule rend les problèmes qui y sont corrélés d'une complexité algorithmique très élevée. Par exemple, tester si un ensemble de clauses fait partie de l'ensemble des MUS d'une instance est un problème Σ_2^P-difficile (Eiter et Gottlob 2002). Bien que théoriquement très difficile, en pratique plusieurs voies ont déjà été explorées avec succès. Parmi ces travaux, une approche proposée par Grégoire *et al.* (2009b) s'appuie sur un parcours stochastique de l'espace de recherche [5] et les propriétés 1.4 et 1.5 afin d'estimer de manière heuristique quelles sont les clauses qui ont une forte chance d'appartenir à un MUS. La première propriété, proposée par Mazure *et al.* (1998), s'appuie sur le fait qu'un MUS est insatisfiable et donc que, quelle que soit l'interprétation $\mathcal{I}$ considérée, $\mathcal{I}$ falsifie une clause de ce MUS.

Propriété 1.4. *Soient Σ une formule* CNF *insatisfiable et Γ un* MUS *de Σ, alors nous avons que :*

$$\forall \mathcal{I}, \exists \alpha \in \Gamma \text{ telle que } \mathcal{I} \not\models \alpha$$

où $\mathcal{I}$ est une interprétation complète construite sur les variables de Σ.

4. Pour déterminer si un ensemble de clauses Σ est un MUS, il suffit de tester $\forall \alpha \in \Sigma$ (nombre polynomial) si $\Sigma \setminus \{\alpha\}$ est satisfiable (appel à un oracle NP).
5. Ce parcours est basé sur la recherche locale qui est présentée dans le chapitre 2.

La seconde propriété, proposée par Grégoire *et al.* (2006), étend la propriété précédente afin de prendre en compte le voisinage de l'interprétation pour savoir si une clause doit être considérée ou non. Les clauses ainsi caractérisées sont dites critiques et sont définies formellement de la manière suivante :

Définition 1.42 (Clause critique)**.** *Soient Σ une formule* CNF *et $\mathcal{I}$ une interprétation complète construite sur les variables de Σ. Une clause $\alpha \in \Sigma$ est dite critique si et seulement si $\mathcal{I}(\alpha) = \bot$ et $\forall \ell \in \alpha, \exists \beta \in \Sigma$ telle que $\tilde{\ell} \in \beta$ et β est unisatisfaite par $\mathcal{I}$. Ces clauses unisatisfaites relatives à une clause critique α sont dites liées à α.*

Exemple 1.16. *Soient Σ la formule de l'exemple 1.15 et l'interprétation complète $\mathcal{I} = \{a, \neg b, c, \neg d, e\}$, la clause $(b \vee \neg c)$ est critique et les clauses $(\neg b)$ et $(\neg a \vee c \vee d)$ sont liées à $(b \vee \neg c)$.*

Les clauses critiques présentent des propriétés intéressantes quant à l'extraction de MUS. En particulier, la propriété suivante garantit l'existence pour chaque clause appartenant à l'ensemble des MUS, d'une interprétation qui la rende critique.

Propriété 1.5. *Soit Γ un* MUS, *$\forall \in \alpha \in \Gamma$ il existe une interprétation complète construite sur les variables de Γ qui rend critique α.*

Cette dernière propriété clôture notre présentation du problème SAT. Dans la section suivante nous présentons un autre problème de décision NP-complet tout aussi important : le problème CSP.

1.3 Le problème CSP

Le problème de satisfaction de contraintes (CSP pour *Constraint Satisfaction Problem*), est, à l'instar de SAT, un problème de décision NP-complet. Un grand nombre de problèmes en intelligence artificielle et d'autres domaines en informatique peuvent être considérés comme des cas particuliers de problèmes CSP (Nadel 1990, Buscemi et Montanari 2008). Ces derniers sont généralement présentés sous la forme d'un ensemble de variables (auxquelles sont associés des domaines de valeurs) et d'un ensemble de contraintes, le tout formant un réseau de contraintes. L'objectif du problème CSP est alors de déterminer s'il existe une assignation des variables telle que toutes les contraintes soient satisfaites.

Dans ce qui suit, nous rappelons les notations et définitions liées au formalisme CSP ainsi qu'un exemple de son utilisation (le problème des n-reines). Pour terminer, tout comme pour le problème SAT, nous présentons le concept d'ensemble minimalement inconsistant de contraintes (ou MUC) d'un CSP.

1.3.1 Formalisation CSP

La programmation par contraintes a été développée dès les années 70 afin de formaliser et résoudre informatiquement de nombreux problèmes. Cette formalisation s'appuie sur la construction d'un réseau de contraintes permettant de modéliser le problème à résoudre.

1.3.1.1 Définition d'un réseau de contraintes

Les réseaux de contraintes sont les fondations sur lesquelles reposent la représentation des problèmes CSP. Ils permettent la formalisation des problèmes de satisfaction de contraintes par une représentation à base de variables et de contraintes. Chaque contrainte est définie sur un sous-ensemble de l'ensemble des variables et limite les combinaisons de valeurs que peuvent prendre ces variables.

Définition 1.43 (réseau de contraintes)**.** *Un réseau de contraintes $\mathcal{P}$ est défini par un couple $\langle \mathcal{X}, \mathcal{C} \rangle$ où :*
- *$\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$ est l'ensemble fini des variables du problème tel qu'à chaque variable X_i est associé un domaine fini $dom(X_i)$ indiquant les valeurs autorisées pour X_i ;*
- *$\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ est un ensemble de m contraintes. Chaque contrainte $C_i \in \mathcal{C}$ est définie par un couple $(var(C_i), rel(C_i))$ où :*
  - *$var(C_i)$ est un ensemble de variables $\{X_{i_1}, X_{i_2}, \ldots, X_{i_{n_i}}\} \subseteq \mathcal{X}$ sur lesquelles porte la contrainte C_i ;*
  - *$rel(C_i)$ est une relation, définie par un sous-ensemble du produit cartésien $\prod_{j=1}^{n_i} dom(X_{i_j})$ des domaines associés aux variables de $var(C_i)$. Ces relations peuvent être exprimées sous différentes formes : table de valeurs compatibles, formules mathématiques, etc.*

Pour écrire un tel réseau de manière formelle il est nécessaire de se fixer un langage. Ce langage est le langage du premier ordre, dans lequel nous retrouvons naturellement les symboles des variables (x, y, x_1, X, etc), des fonctions (arithmétique ($+, -, \times, /$) ou booléennes ($\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$)) et des relations ($=, <, >, \leq, \geq$) .

Exemple 1.17. *Nous définissons un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ avec :*
- *$\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5\}$;*
- *$dom(X_1) = dom(X_2) = dom(X_3) = dom(X_4) = dom(X_5) = \{0, 1\}$;*
- *$\mathcal{C} = \{C_1, C_2, C_3\}$ tel que $C_1 = (X_1 + X_2 < X_4)$, $C_2 = (X_4 \neq X_5)$ et $C_3 = (X_3 = X_5)$.*

Comme nous pouvons le remarquer, les contraintes d'un réseau ne portent pas toutes sur le même nombre de variables. Ce nombre appelé arité de la contrainte est défini comme suit :

Définition 1.44 (arité)**.** *L'arité d'une contrainte $C \in \mathcal{C}$ est le nombre de variables sur lesquelles elle porte, c'est-à-dire $|var(C)|$. Dans le cas où cette arité vaut :*
- *1, la contrainte est dite unaire ;*
- *2, la contrainte est dite binaire ;*
- *$n > 2$, la contrainte est dite n-aire.*

Exemple 1.18. *Les contraintes C_1 et C_2 du réseau de contraintes de l'exemple 1.17 sont respectivement n-aire et binaire.*

La contrainte d'arité maximale d'un réseau de contraintes permet de définir sa nature. Si toutes les contraintes d'un réseau de contraintes $\mathcal{P}$ sont binaires, $\mathcal{P}$ est lui-même dit binaire. Réciproquement, lorsqu'au moins une contrainte est d'arité supérieure à deux, le réseau de contraintes est dit non-binaire (n-aire). Un réseau de contraintes non-binaires peut toujours être transformé en un réseau de contraintes binaires en appliquant une technique de binarisation. Les deux méthodes les plus utilisées sont l'encodage du graphe dual (*dual graph encoding*) (Dechter et Pearl 1987) et l'encodage des variables cachées (*hidden variable encoding*) (Rossi *et al.* 1990).

Un réseau de contraintes binaires (respectivement n-aires) est souvent représenté comme un graphe de contraintes (respectivement hypergraphe de contraintes). Ce graphe de contraintes permet d'avoir un rendu graphique du problème formalisé.

Définition 1.45 (graphe et hyper-graphe de contraintes)**.** *À chaque réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ peut être associé, pour les contraintes binaires, un graphe des contraintes obtenu en représentant chaque variable du réseau par un sommet et chaque contrainte binaire $C \in \mathcal{C}$ qui porte sur les variables X_i et X_j par une arête entre les sommets X_i et X_j. Dans le cas des réseaux de contraintes n-aires, une représentation par des hyper-graphes est utilisée, en remplaçant les arêtes par des hyper-arêtes.*

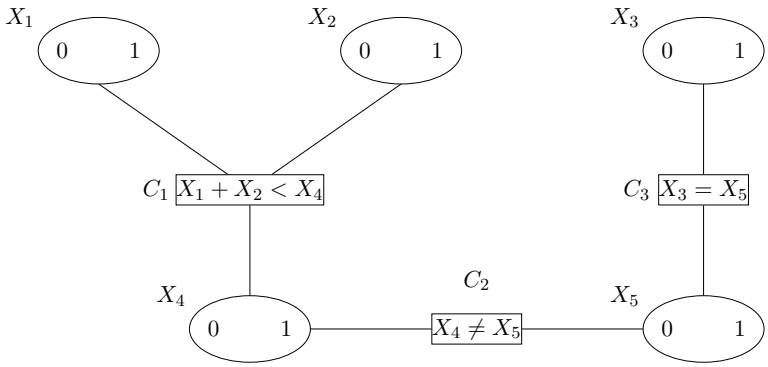


FIGURE 1.5 – Représentation graphique d'un CSP.

Exemple 1.19. *La figure 1.5 donne une représentation du réseau de contraintes de l'exemple 1.17.*

Comme nous l'avons souligné précédemment, les contraintes peuvent être exprimées sous différentes formes. Une contrainte C peut être définie comme un ensemble de n-uplets de valeurs si l'on considère un ordre sur les variables de sa portée. Cet ordre arbitraire permet de définir la relation $rel(C)$ comme un sous-ensemble du produit cartésien des domaines des variables appartenant à $var(C)$, représentant alors les n-uplets autorisés (interdits) par la contrainte C.

Définition 1.46 (tuples autorisés et interdits)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contrainte et $C_i \in \mathcal{C}$ telle que $var(C) = \{X_{i_1}, X_{i_2}, \ldots, X_{i_{n_i}}\}$. Un tuple autorisé (respectivement interdit) est un élément de $\prod_{j=1}^{n_i} dom(X_{i_j})$ tel que la relation $rel(C_i)$ est satisfaite (respectivement falsifiée). L'ensemble des tuples autorisés (respectivement interdits) est noté $\mathfrak{T}_a$ (respectivement $\mathfrak{T}_i$).*

Remarque 1.8. *Lorsqu'aucune information n'est fournie, il est admis que l'ordre utilisé sur les variables est l'ordre lexicographique.*

Exemple 1.20. *Considérons la contrainte C_2 de l'exemple 1.17 portant sur les variables X_4 et X_5 de domaine $\{0, 1\}$. Pour l'ordre lexicographique ($X_4 < X_5$), la contrainte $C_2 = (X_4 \neq X_5)$ peut être définie à l'aide de l'ensemble des tuples autorisés de la manière suivante : $C_2 = (\{X_4, X_5\}, \{(0, 1), (1, 0)\})$.*

Remarque 1.9. *Lorsqu'un réseau de contraintes est entièrement défini à l'aide de tuples, il est dit représenté en* extension*. Dans le cas contraire, il est dit représenté en* intension*.*

Un réseau de contraintes peut également être représenté graphiquement par sa micro-structure. Dans ce cas, un graphe n-parti (pour un réseau à n variables) est dit de compatibilité où un sommet représente un couple (variable, valeur), et une hyper-arête un n-uplet autorisé par l'une des contraintes du problème. Il est également possible d'exploiter le graphe d'incompatibilité plutôt que le graphe de compatibilité. Il s'agit de la micro-structure complémentaire, c'est-à-dire le graphe dual où une arête n'encode pas un support mais un conflit entre des valeurs. De telles représentations sont notamment utilisées afin de détecter (et éliminer) des symétries au niveau du problème (Gent et Smith 1999, Fahle *et al.* 2001) ou encore dans le développement de méthodes de décomposition de problèmes (Chmeiss *et al.* 2003). La figure 1.6 donne la représentation de la micro-structure du réseau de contraintes de l'exemple 1.17.

Un réseau de contraintes peut être caractérisé par plusieurs grandeurs : le nombre de variables et la taille maximale de leurs domaines, le nombre de contraintes. Nous utilisons dans la suite de ce manuscrit les notations suivantes :

– n : le nombre de variables du problème ;

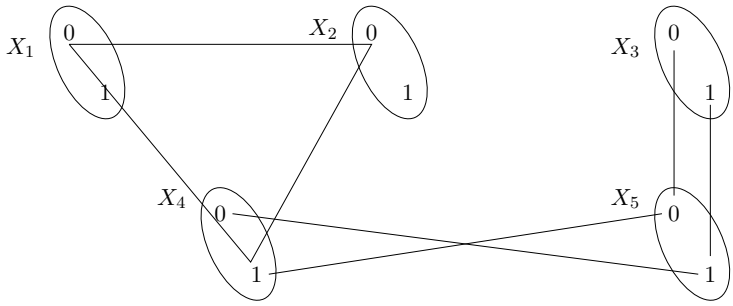


FIGURE 1.6 – Représentation de la microstructure d'un réseau de contraintes.

– d : la taille du plus grand domaine ;
– e : le nombre de contraintes ;
– r : l'arité maximale des contraintes.

Nous donnons dans ce qui suit un exemple de modélisation d'un problème concret sous la forme d'un réseau de contraintes.

1.3.1.2 Un exemple de représentation : Le problème des n-reines

Le problème des n-reines est un problème « jouet », proposé initialement en 1848 par un joueur d'échec du nom de Max Bassel. Ce problème consiste à disposer n reines sur un échiquiers $n \times n$ de manière à ce qu'aucune d'entre elles ne soit en prise [6] avec les autres.

Ce problème se formalise naturellement à l'aide d'un réseau de contraintes. En effet, en exploitant le fait qu'une seule reine est placée par colonne, le problème se réduit au choix de la ligne. En conséquence de quoi, une modélisation du problème considère chaque colonne i comme une variable X_i de domaine $dom(X_i) = \{1, 2, \ldots, n\}$ où chaque valeur de X_i désigne le numéro de ligne où se place la reine. Les contraintes quant à elles doivent retranscrire l'énoncé du problème, c'est-à-dire que deux reines ne doivent jamais se retrouver sur la même ligne ($X_i \neq X_j$) ou la même diagonale ($|X_i + i - j| \neq X_j$ et $|X_i - i + j| \neq X_j$). Si nous nous plaçons dans le cadre général avec n-reines, nous obtenons le réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ suivant :

– $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$;
– $dom(X_1) = dom(X_2) = \cdots = dom(X_n) = \{1, 2, \ldots, n\}$;
– $\mathcal{C}$ est défini par, $\forall i, j : (X_i \neq X_j) \wedge (|X_i - X_j| \neq i)$.

La figure 1.7 illustre le réseau de contraintes associé au problème des 4-reines.

La résolution de ce réseau consiste à affecter des valeurs aux variables, de telle sorte que toutes les contraintes soient satisfaites. Avant de pouvoir résoudre un tel problème, il est nécessaire de définir les notions d'instanciation, d'interprétation, d'assignation, de réfutation et de cohérence d'un réseau de contraintes, qui sont introduites dans la partie suivante.

1.3.1.3 Instanciation, interprétation et cohérence

À partir de la modélisation d'un problème sous la forme d'un réseau de contraintes, nous souhaitons vérifier si celui possède ou non une solution. Il est donc nécessaire de définir la sémantique d'un réseau

6. Deux reines sont en prise si elles se trouvent sur une même colonne, une même ligne ou une même diagonale de l'échiquier.

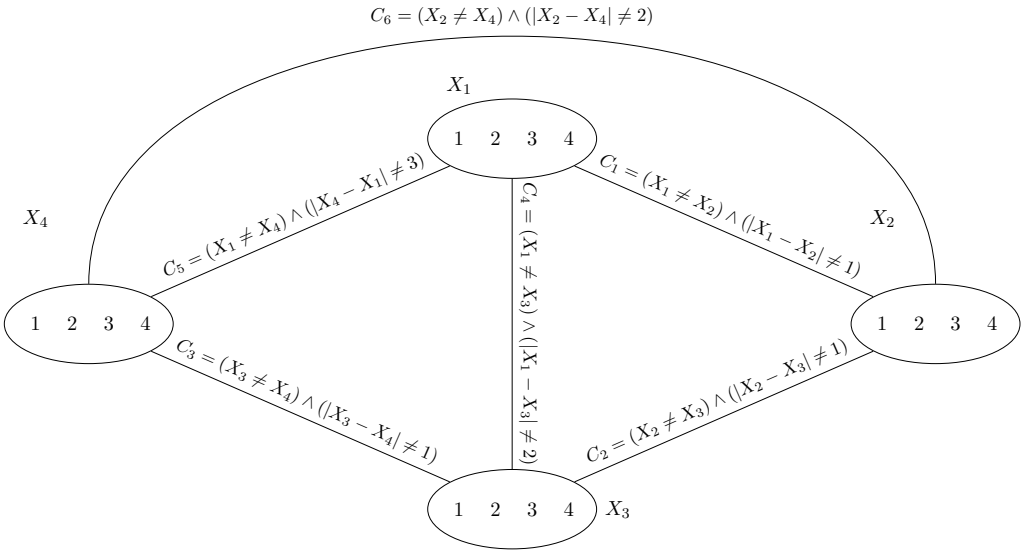


FIGURE 1.7 – Problème des 4-reines et le réseau de contrainte associé.

de contraintes, c'est-à-dire dans quelles conditions ce réseau est satisfait. Afin d'obtenir une réponse à cette question, il est nécessaire d'introduire certaines notions, telles que l'association d'une valeur à une variable ou la satisfaction d'une contrainte. Tout d'abord, nous caractérisons l'association d'une valeur à une variable.

Définition 1.47 (instanciation)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, une instanciation $\mathcal{A}$ de $\mathcal{Y} \subseteq \mathcal{X}$, où $\mathcal{Y} = \{X_{y_1}, X_{y_2}, \ldots X_{y_{|\mathcal{Y}|}}\}$, est une application qui associe à chaque variable X_{y_i} une valeur $\mathcal{A}(X_{y_i}) \in dom(X_{y_i})$. Nous notons $\mathcal{A}(\mathcal{Y})$ l'ensemble de valeurs assignées par $\mathcal{A}$ aux variables de $\mathcal{Y}$.*

Exemple 1.21. *Considérons l'exemple des 4-reines présenté précédemment (voir §1.3.1.2). Nous avons $\mathcal{A} = \{(X_1 = 1), (X_2 = 4), (X_3 = 1)\}$ une instanciation de $\mathcal{X}$ et $\mathcal{A}(\{X_1, X_2\}) = \{X_1 = 1, X_2 = 4\}$.*

Définition 1.48 (instanciation partielle, complète et incomplète)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, une instanciation $\mathcal{A}$ est dite partielle, complète ou incomplète si $\mathcal{A}$ est définie sur un ensemble de variables $\mathcal{Y}$ tel que nous avons respectivement $\mathcal{Y} \subseteq \mathcal{X}$, $\mathcal{Y} = \mathcal{X}$ et $\mathcal{Y} \subset \mathcal{X}$.*

Exemple 1.22. *Soit le réseau de contraintes $\mathcal{P}$ de l'exemple des 4-reines, $\mathcal{A}_1 = \{(X_1 = 1), (X_3 = 1)\}$ et $\mathcal{A}_2 = \{(X_1 = 1), (X_2 = 2), (X_3 = 1), (X_4 = 3)\}$ sont respectivement des instanciations incomplètes et complètes. Ces deux instanciations sont aussi partielles.*

Nous définissons à présent quelles sont les conditions pour qu'une instanciation satisfasse un ensemble de contraintes.

Définition 1.49 (contrainte satisfaite et falsifiée)**.** *Soient un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$, une instanciation $\mathcal{A}$ de $\mathcal{Y} \subseteq \mathcal{X}$ et une contrainte $C \in \mathcal{C}$ telle que $var(C) \subseteq \mathcal{Y}$. L'instanciation $\mathcal{A}$ satisfait (respectivement falsifie) la contrainte C, noté $\mathcal{A} \models C$ (respectivement $\mathcal{A} \not\models C$), si et seulement si $\mathcal{A}(var(C)) \in rel(C)$ (respectivement $\mathcal{A}(var(C)) \notin rel(C)$). L'ensemble des contraintes falsifiées par $\mathcal{A}$ est noté $false(\mathcal{P}, \mathcal{A})$.*

Exemple 1.23. *Considérons le réseau de contraintes associé à l'exemple des 4-reines, l'instanciation $\mathcal{A} = \{(X_1 = 1), (X_2 = 3)\}$ satisfait la contrainte C_1.*

Lorsqu'une contrainte est satisfaite quelque soit l'instanciation considérée, cette contrainte est dite universelle. Ce qui se définit formellement comme suit :

Définition 1.50 (contrainte universelle)**.** *Soit un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$, une contrainte $C \in \mathcal{C}$ est dite universelle si et seulement si C est satisfaite par toutes les instanciations de $rel(C)$.*

Étant donné un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$, une instanciation $\mathcal{A}$ d'un sous-ensemble $\mathcal{Y}$ de variables de $\mathcal{X}$ est dite consistante si toutes les contraintes $C \in \mathcal{C}$ dont la portée est incluse dans $\mathcal{Y}$ sont satisfaites.

Définition 1.51 (instanciation consistante)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes et $\mathcal{Y} \subseteq \mathcal{X}$ un sous-ensemble de variables. Une instanciation $\mathcal{A}$ de $\mathcal{Y}$ est consistante si et seulement si $\forall C \in \mathcal{C}$ telle que $var(C) \subseteq \mathcal{Y}$, C est satisfaite par $\mathcal{A}$.*

Une solution d'un réseau consiste donc à déterminer une instanciation complète qui satisfait toutes les contraintes du réseau.

Définition 1.52 (solution d'un réseau de contraintes)**.** *Une solution $\mathcal{A}$ d'un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ est une instanciation des variables de $\mathcal{X}$ telle que $\forall C \in \mathcal{C}$ nous avons $\mathcal{A} \models C$. Dans ce cas $\mathcal{A}$ satisfait $\mathcal{P}$ (noté $\mathcal{A} \models \mathcal{P}$).*

Remarque 1.10. *Dans le cas où une instanciation $\mathcal{A}$ ne satisfait pas un réseau de contraintes $\mathcal{P}$ alors il est dit que $\mathcal{A}$ falsifie $\mathcal{P}$ (noté $\mathcal{A} \not\models \mathcal{P}$).*

Exemple 1.24. *L'instanciation $\mathcal{A} = \{(X_1 = 3), (X_2 = 1), (X_3 = 4), (X_4 = 2)\}$ est une solution du réseau de contraintes des 4-reines défini dans le paragraphe 1.3.1.2. Comme nous pouvons le voir sur la figure 1.8 cette configuration satisfait bien le problème énoncé.*

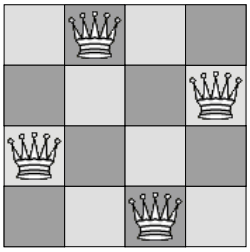


FIGURE 1.8 – Une solution du problème des 4-reines.

Le problème de satisfaction CSP s'énonce alors de la façon suivante.

Définition 1.53 (problème de satisfaction de contraintes)**.** *Le problème CSP est le problème de décision qui consiste à savoir si un réseau de contraintes possède ou pas une solution.*

Remarque 1.11. *Par abus de langage, un réseau de contraintes peut aussi être appelé CSP.*

Comme nous venons de le définir, une instanciation consiste à *assigner* une valeur à une variable. Néanmoins dans le cadre CSP, il est aussi possible de *réfuter* des valeurs du domaine d'une variable [7]. Ces deux opérations se définissent formellement de la manière suivante :

7. Dans le cadre de SAT, supprimer une valeur (ou littéral) du domaine d'une variable conduit irrémédiablement à considérer l'autre.

Définition 1.54 (assignation et réfutation)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $X \in \mathcal{X}$ telle que $|dom(X)| > 0$ et v une valeur du domaine de X. Une assignation $(X = v)$ (respectivement réfutation $(X \neq v)$), notée $\mathcal{P}_{|(x=v)}$ (respectivement $\mathcal{P}_{|(x \neq v)}$), consiste à associer (respectivement supprimer) la valeur v à la variable X, c'est-à-dire à modifier le domaine de X tel que $dom(X) = \{v\}$ (respectivement $dom(X) = dom(X) \setminus \{v\}$).*

Exemple 1.25. *Considérons le réseau de contraintes $\mathcal{P}$ de l'exemple des 4-reines, l'assignation $\mathcal{P}_{|(x=2)}$ et la réfutation de $\mathcal{P}_{|(x \neq 2)}$ entraînent respectivement que $dom(X_1) = \{2\}$ et $dom(X_1) = \{1, 3, 4\}$.*

Remarque 1.12. *Assigner X à v conduit à réfuter l'ensemble des valeurs de X différentes de v.*

À partir de ces deux notions, il est possible de définir une notion plus générale que celle d'instanciation. En effet, il n'est pas rare dans le cadre CSP d'assigner et/ou de réfuter des valeurs lorsque la résolution d'un réseau de contraintes est envisagée. Dans ce cadre, nous parlons d'interprétation et non plus d'instanciation.

Définition 1.55 (interprétation)**.** *Soit $\mathcal{P}$ un réseau de contraintes, une interprétation $\mathcal{I}$ est définie comme un ensemble d'assignations et/ou de réfutations de $\mathcal{P}$. Comme pour SAT, l'ensemble des interprétations est noté $\mathfrak{S}$.*

Exemple 1.26. *Considérons le réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ de l'exemple des 4-reines, $\mathcal{I} = \{(X_1 \neq 1), (X_2 = 2), (X_3 \neq 1), (X_3 \neq 2), (X_3 \neq 3), (X_4 \neq 1)\}$ est une interprétation de $\mathcal{P}$.*

Remarque 1.13. *Une instanciation est une interprétation particulière constituée uniquement d'assignations.*

Notation 1.2. *Soit $\mathcal{I}$ une interprétation, nous notons $\mathcal{I}_{\overline{X}}$ l'interprétation obtenue à partir de $\mathcal{I}$ en supprimant toutes les assignations et réfutations concernant la variable X.*

Nous pouvons étendre la notion d'application (|) d'une assignation ou d'une réfutation au cas d'ensemble, c'est-à-dire au cas d'interprétation.

Définition 1.56 (application d'une interprétation)**.** *Soient un réseau de contraintes $\mathcal{P}$ et une interprétation $\mathcal{I} = \{(X_1 \diamond v_1), (X_2 \diamond v_2), \ldots, (X_n \diamond v_n)\}$ telle que $(X_i \diamond v_i)$ représente aussi bien une assignation qu'une réfutation. Le réseau de contraintes obtenu par l'application de l'interprétation est noté $\mathcal{P}_{|\mathcal{I}}$ et est calculé inductivement de la manière suivante $(((\mathcal{P}_{|(X_1 \diamond v)})_{|(X_2 \diamond v_2)})_{|\ldots})_{|(X_n \diamond v_n)}$.*

Exemple 1.27. *Considérons le réseau de contraintes $\mathcal{P}$ de l'exemple des 4-reines et l'interprétation $\mathcal{I}$ de l'exemple 1.26. Le réseau de contraintes $\mathcal{P}_{|\mathcal{I}}$ est tel que $dom(X_1) = \{2, 3, 4\}$, $dom(X_2) = \{2\}$, $dom(X_3) = \{4\}$ et $dom(X_1) = \{2, 3, 4\}$.*

Il est possible à partir d'une interprétation $\mathcal{I}$ d'extraire une instanciation. Pour cela, il suffit de considérer le réseau de contraintes obtenu par l'application de $\mathcal{I}$ et d'assigner les variables *singletons*, c'est-à-dire les variables ne possédant plus qu'une seule valeur dans leur domaine.

Définition 1.57 (instanciation d'une interprétation)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $\mathcal{I}$ une interprétation construite sur les variables de $\mathcal{P}$ et $\mathcal{P}_{|\mathcal{I}} = \langle \mathcal{X}', \mathcal{C} \rangle$ où $\mathcal{X}'$ est l'ensemble des variables obtenu après l'application de l'interprétation $\mathcal{I}$ sur le réseau $\mathcal{P}$. L'instanciation obtenue à partir de $\mathcal{I}$ est $\mathcal{A}^{\mathcal{I}} = \{(X = v)$ tel que $X \in \mathcal{X}'$ et $dom(X) = \{v\}\}$.*

Exemple 1.28. *Soient $\mathcal{P}$ le réseau de contraintes de l'exemple des 4-reines et $\mathcal{I}$ l'interprétation de l'exemple 1.26, l'instanciation obtenue à partir de $\mathcal{I}$ est $\mathcal{A}^{\mathcal{I}} = \{(X_2 = 2), (X_3 = 4)\}$.*

Cette définition nous permet d'étendre les notions d'instanciation complète, partielle, incomplète et consistante ainsi que les notations pour une interprétation.

Définition 1.58 (interprétation complète, partielle, incomplète, modèle et consistante)**.** *Soient un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ et une interprétation $\mathcal{I}$ construite sur les variables de $\mathcal{P}$. L'interprétation $\mathcal{I}$ est dite partielle, complète, incomplète consistante ou modèle si et seulement si l'instanciation $\mathcal{A}^{\mathcal{I}}$ obtenue à partir de $\mathcal{I}$ est respectivement partielle, complète, incomplète, consistante ou modèle.*

Notation 1.3. *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $\mathcal{I}$ une interprétation. Nous notons $\mathcal{I} \models \mathcal{P}$ (respectivement $\mathcal{I} \not\models \mathcal{P}$) lorsque $\mathcal{I}$ est un modèle (respectivement contre-modèle) de $\mathcal{P}$.*

Il est probable qu'un réseau de contraintes soit directement *incohérent*, c'est-à-dire que l'assignation des variables singleton du réseau produit une interprétation inconsistante.

Définition 1.59 (réseau directement cohérent et incohérent)**.** *Soit $\mathcal{P}$ un réseau de contraintes, $\mathcal{P}$ est directement cohérent si et seulement si l'instanciation obtenue à partir de l'interprétation vide ($\mathcal{I} = \emptyset$) est consistante. Dans le cas contraire, le réseau de contraintes $\mathcal{P}$ est dit incohérent.*

Exemple 1.29. *Soient le réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ de l'exemple de 4-reines et les deux sous-réseaux $\mathcal{P}_1$ et $\mathcal{P}_2$ obtenus respectivement par simplification à partir des deux interprétations $\mathcal{I}_1 = \{(X_1 = 1)\}$ et $\mathcal{I}_2 = \{(X_1 = 1), (X_2 \neq 1), (X_2 \neq 3), (X_2 \neq 4)\}$. Puisque $\mathcal{A}^{\mathcal{I}_1} = \{(X_1 = 1)\}$ et $\mathcal{A}^{\mathcal{I}_2} = \{(X_1 = 1), (X_2 = 2)\}$ les réseaux de contraintes $\mathcal{P}_1$ et $\mathcal{P}_2$ sont respectivement directement cohérents et directement incohérents.*

Comme pour le problème SAT, il est parfois intéressant, lorsqu'un réseau de contraintes ne possède pas de solution, de pouvoir déterminer l'ensemble des contraintes responsables de l'incohérence. La détection de cet ensemble, appelé noyau minimalement incohérent ou MUC, est présentée dans la partie suivante.

1.3.2 Noyaux minimalement inconsistants (MUC)

Le problème de la détection de noyaux minimalement inconsistants dans le cadre CSP est équivalent à la détection de MUS dans le cadre SAT. Comme pour le problème SAT, tout CSP $\mathcal{P}$ incohérent admet au moins un sous-réseau $\mathcal{P}'$ de $\mathcal{P}$ insatisfiable de taille minimale, c'est-à-dire tel que chaque sous-réseau $\mathcal{P}''$ de $\mathcal{P}'$ est satisfiable. En pratique, extraire de telles informations lorsqu'une inconsistance est constatée dans un système permet d'identifier les éléments qui sont en conflits et peut aider à comprendre, expliquer, diagnostiquer les causes et permet de proposer un système rendu consistant. Les éléments considérés en conflits sont généralement représentés par un sous-ensemble de contraintes du problème. Ce sous-ensemble de contraintes, appelé MUC, est défini formellement de la manière suivante :

Définition 1.60 (Formule Minimalement Inconsistante)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes insatisfiables. Le réseau de contraintes $\mathcal{P}' = \langle \mathcal{X}, \mathcal{C}' \rangle$ est un* MUC *(Minimally Unsatisfiable Core) de $\mathcal{P}$ si et seulement si :*

- *$\mathcal{C}' \subseteq \mathcal{C}$;*
- *$\mathcal{P}'$ est insatisfiable ;*
- *$\forall \mathcal{C}'' \subset \mathcal{C}$, le réseau de contraintes $\langle \mathcal{X}, \mathcal{C}'' \rangle$ est satisfiable.*

Exemple 1.30. *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes tel que :*

- *$\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5\}$;*
- *$dom(X_1) = dom(X_2) = dom(X_3) = dom(X_4) = dom(X_5) = \{1, 2, 3\}$;*

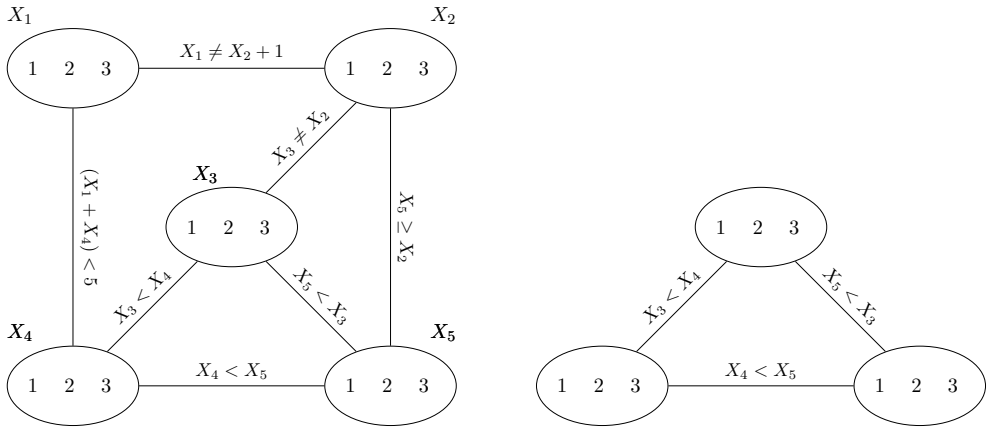


FIGURE 1.9 – Réseau de contraintes incohérent avec l'un de ses MUC.

– $\mathcal{C} = \{X_1 \neq X_2 + 1, X_5 \geq X_2, X_5 < X_3, X_4 < X_5, X_3 < X_4, X_3 \neq X_2, (X_1 + X_4) < 5\}$.
La représentation graphique de $\mathcal{P}$ et de l'un de ces MUC *sont illustrées dans la figure 1.9.*

L'identification d'une source d'incohérence d'un problème par un MUC permet d'en restaurer localement la cohérence par la suppression d'une contrainte de cet ensemble. En effet, dans l'exemple précédent, retirer n'importe quelle contrainte du MUC est suffisant pour obtenir un sous-problème satisfiable. Malheureusement, comme pour le problème SAT, le nombre de MUC est exponentiel en nombre de contraintes et de manière générale la restauration de la cohérence d'un réseau passe par la « réparation » de chacun de ses MUC.

De la même manière que pour le problème de la détection de MUS, plusieurs approches pour la résolution pratique de ce problème ont été proposées. Ces approches s'appuient pour la plupart sur une pondération des contraintes apparaissant le plus souvent dans les conflits rencontrés lors du processus de recherche d'une solution [8]. Une autre approche consiste, comme pour la détection de MUS, à s'appuyer sur un parcours stochastique de l'espace de recherche [9] afin d'estimer de manière heuristique quelles sont les contraintes appartenant aux MUC. Cette approche, expérimentée par Eisenberg et Faltings (2003), s'appuie sur la même propriété [10] qu'énoncée dans le cadre de SAT.

Propriété 1.6. *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes insatisfiable et $\mathcal{P}' = \langle \mathcal{X}, \mathcal{C}' \rangle$ un* MUC *de $\mathcal{P}$, alors nous avons que :*

$$\forall \mathcal{I}, \exists C \in \mathcal{C}' \text{ telle que } \mathcal{I} \not\models C$$

où $\mathcal{I}$ est une interprétation complète définie sur $\mathcal{X}$.

Dans les deux dernières sections, nous avons présenté deux formalismes pour la modélisation et la résolution de problèmes sous contraintes. Afin de résoudre les problèmes ainsi générés il est nécessaire de mettre au point différents algorithmes. Néanmoins, il paraît évident que la nature même du problème conditionne les performances des algorithmes proposés. En effet, un algorithme dédié à un problème a de fortes chances d'être meilleur qu'un algorithme générique sur ce même problème. Afin de tester la robustesse des algorithmes, un grand nombre d'instances de problèmes sont donc utilisées. Ces différentes instances sont présentées dans la section suivante.

8. Ces processus de résolution sont présentés dans le chapitre 3.
9. Ce parcours est effectué sur la base d'un algorithme de recherche locale (voir chapitre 2)
10. La preuve de cette propriété est similaire à celle proposée par Mazure *et al.* (1998)

1.4 Les instances

Les performances des différents algorithmes proposés afin de résoudre le problème SAT ou CSP sont en général évaluées et comparées sur un ensemble de formules ou de réseaux de contraintes. Ces dernières sont classiquement divisées en deux catégories : les instances aléatoires et les instances structurées. Chaque algorithme présente souvent un comportement différent vis-à-vis de la catégorie d'instances utilisée. La notion d'instances difficiles se révèle alors complexe. Certaines caractéristiques permettent néanmoins de déterminer de manière probabiliste si une instance est difficile à résoudre, sans pouvoir toutefois assurer qu'elle l'est pour tous les algorithmes.

1.4.1 Instances aléatoires

L'évaluation empirique des différents algorithmes proposés pour la résolution des problèmes SAT et CSP nécessite de pouvoir comparer leurs performances sur des problèmes possédant des caractéristiques bien définies. Pour effectuer cela, le moyen le plus simple est de générer des instances de manière aléatoire. De telles instances sont fournies par des générateurs qui leur assurent des caractéristiques rendant leur résolution plus difficile. Leur génération est souvent très simple et contrôlée par un nombre limité de paramètres. De tels générateurs ont des propriétés intéressantes permettant la génération d'instances non triviales à résoudre. Parmi ces propriétés, la transition de phase est sans doute la plus étudiée.

Ce terme, initialement utilisé en physique, est employé pour décrire un changement abrupt et soudain provoqué par le changement de paramètres extérieurs. Cette transition a lieu lorsque certains paramètres atteignent un certain seuil. Ce phénomène de seuil est observé quand la probabilité pour une propriété d'être satisfaite passe de presque 0 à presque 1. Le seuil est caractérisé par les valeurs des paramètres qui délimitent la zone de transition pour des instances de tailles infinies.

Pour les problèmes de satisfaction de contraintes tel que SAT et CSP, ces paramètres sont le plus souvent liés au nombre et à la nature des contraintes du problème. Des contraintes trop dures ou trop nombreuses, ou au contraire trop lâches ou éparses rendent le problème respectivement trivialement insatisfiable (sur-contraint) ou satisfiable (sous-contraint). Néanmoins, il existe un zone intermédiaire où les instances générées ont une probabilité de $\frac{1}{2}$ d'être satisfiables (ou insatisfiables). C'est souvent dans cette zone, dite « critique », que se situent les instances les plus difficiles à résoudre.

1.4.1.1 Générateur d'instances aléatoires au seuil pour SAT

Le modèle de génération d'instances aléatoires classiquement utilisé dans le cadre de SAT est basé sur la construction d'instances k-SAT. De telles instances sont caractérisées par trois paramètres : le nombre de variables V, le nombre de clauses C et la taille des clauses k.

Le générateur standard d'instances aléatoires k-SAT consiste à générer des problèmes où toutes les clauses sont de taille fixe k. La génération d'une clause se fait en tirant uniformément k littéraux distincts parmi l'ensemble des $2 \times n$ littéraux du problème.

Le phénomène de seuil pour ce générateur d'instances a été mis en avant par les travaux de Mitchell *et al.* (1992) et complété par ceux de Monasson *et al.* (1999). Ils montrent que le rapport entre le nombre de clauses et le nombre de variables ($\frac{C}{V}$) permet de donner des indications sur la satisfiabilité de l'instance. Pour chaque valeur de k, un seuil a été empiriquement calculé. La classe d'instances la plus étudiée étant la classe 3-SAT, beaucoup de travaux ont permis d'approximer finement la valeur du seuil de difficulté (Achlioptas 2009). Ce dernier est à l'heure actuelle estimé expérimentalement proche de $\frac{C}{V} = 4.25$.

1.4.1.2 Générateur d'instances aléatoires au seuil pour CSP

De manière générale, pour le problème CSP, un problème aléatoire est défini selon un quintuplet $\langle k, n, d, e, t\rangle$, donnant respectivement l'arité des contraintes, le nombre de variables, la taille des domaines, le nombre et la dureté des contraintes. Cette dernière est définie, pour une contrainte C, comme le rapport entre le nombre de tuples interdits et le nombre d'interprétations possibles définies sur $var(C)$. La génération d'une instance aléatoire consiste alors à considérer un ensemble de contraintes sélectionnées aléatoirement parmi toutes les contraintes possibles (correspondant aux éléments du produit cartésien des variables entre elles pour lesquels toutes les variables sont distinctes). Les tuples autorisés par chaque contrainte sont ensuite obtenus à partir d'un tirage probabiliste dépendant de t.

Parmi les différents modèles proposés pour la génération de telles instances, le modèle RB, proposé par Xu et Li (2000), est particulièrement intéressant, puisqu'il permet simplement d'obtenir des instances aléatoires au seuil, et éventuellement de les forcer à être satisfiables (Xu *et al.* 2005). Ce générateur définit un problème aléatoire selon le quintuplet $\langle k, n, \alpha, r, t\rangle$ tel que α détermine la taille des domaines $(d = n^{\alpha})$ et r le nombre de contraintes $(e = r \times n \times ln(n))$. Le seuil de difficulté est alors obtenu pour $t = 1 - exp(-\frac{\alpha}{r})$ et est garanti en particulier pour $\alpha > \frac{1}{k}$ et $t < \frac{k-1}{k}$.

1.4.2 Instances structurées

Contrairement aux instances aléatoires, les instances structurées n'ont pas réellement de caractéristiques permettant de prévoir leurs difficultés. Généralement, ces instances sont issues de problèmes industriels (instances industrielles) tels que les problèmes de vérification de circuits intégrés (Velev et Bryant 2003), *Bounded Models Checking* (Biere *et al.* 1999b) et planification (Kautz et Selman 1996) ou de problèmes académiques (instances faites main) tels que les problèmes de coloriation de graphes des pigeons et des n-reines. Ces instances sont donc créées en codant le problème initial comme un problème de satisfaction de contraintes SAT ou CSP.

1.4.3 Instances utilisées

Comparer les différents solveurs entre eux est une tâche difficile. Les résultats obtenus dépendent de la manière dont ont été encodés les problèmes ainsi que des méthodes utilisées pour les résoudre. Il est très difficile de déterminer à l'avance les performances d'un algorithme sur un problème donné. De manière à encourager la recherche d'algorithmes toujours plus performants, des compétions sont organisées (SAT *Race*, SAT *Competition* et CSP *Competition*). Ces compétitions consistent à exécuter un ensemble d'algorithmes pendant un temps limité sur un large panel de problèmes. Dans la suite de ce manuscrit, lorsque des expérimentations sont présentées, les instances utilisées sont issues de ces différentes compétitions.

1.5 Conclusion

Dans ce chapitre nous avons abordé deux problèmes NP-Complets qui sont : le problème SAT et le problème CSP. Ces derniers consistent à trouver une interprétation des variables telle que toutes les contraintes du problème soient satisfaites. Puisqu'il paraît évident que le choix de l'instance étudiée conditionne les performances des algorithmes, nous avons présenté deux catégories d'instances permettant de les comparer : les instances aléatoires et les instances structurées (faites main et industrielles). Dans les prochains chapitres nous présentons plusieurs paradigmes pour la résolution de ces différentes instances.

Chapitre 2

Approches incomplètes : la recherche locale

Sommaire

LES APPROCHES INCOMPLÈTES pour la résolution pratique du problème SAT ou CSP, sont en générales incapables de répondre à l'insatisfaibilité d'une formule. Ces approches effectuent le plus souvent un parcours de l'espace de recherche non systématique pendant un temps donné. Une fois ce temps imparti écoulé, deux cas peuvent se présenter : si un modèle est trouvé, alors la procédure stoppe son exécution et retourne le modèle obtenu, sinon la méthode retourne qu'elle n'a pas trouvé de solution et dans ce cas il est impossible de conclure. En effet, quelle que soit la quantité de temps laissée à la méthode, ce type d'approche ne garantissant pas un parcours exhaustif de l'ensemble des interprétations de la formule, il est impossible d'affirmer qu'il n'y a pas de solution.

Il existe différents types d'approches incomplètes, les principales étant les techniques algorithmiques « génétiques » à base de population (De Jong et Spears 1989, Hao et Raphaël 1994, Holland 1992), la « *survey propagation* » (Braunstein *et al.* 2005), la recherche à voisinage variable (Hansen *et al.* 2001), les algorithmes de colonies de fourmis (Dorigo et Stützle 2004), les algorithmes de parcours réduit de l'arbre de recherche (Ow et Morton 1988, Sourd et Chrétienne 1999) et la recherche locale. Dans ce chapitre seules les techniques basées sur la recherche locale sont présentées, celles-ci étant certainement les plus connues de toutes les méthodes incomplètes pour la résolution des problèmes SAT et CSP.

La recherche locale est une technique venant de la recherche opérationnelle où elle a montré son efficacité sur un grand nombre de problèmes d'optimisations (problème du voyageur de commerce, sac à dos, etc.). Un des inconvénients majeurs de cette approche est qu'elle est incapable de répondre dans le cas où la formule considérée est insatisfiable et cela même en considérant un temps infini. Néanmoins, elle permet, dans des temps de calcul raisonnable, de trouver des interprétations, pas nécessairement modèles de la formule, pouvant être de « bonne » qualité, c'est-à-dire falsifiant peu de contraintes. Elle se distingue en cela des approches exactes, qui garantissent certes la résolution du problème mais souvent au prix de temps de calculs prohibitifs. Il est aussi important de souligner que sur les instances aléatoires au seuil satisfiables, la recherche locale est incontestablement la meilleure méthode de résolution.

Après quelques définitions et l'introduction de l'algorithme de base de la recherche locale, un certain nombre de stratégies d'échappement pour la résolution des problèmes de satisfaction de contraintes SAT et CSP sont présentées. Ces stratégies étant souvent tributaires de plusieurs paramètres, nous présentons une méthode introduite par Hoos (2002) permettant de les régler de manière dynamique.

2.1 Principe

Les techniques à base de recherche locale ont été développées initialement afin de résoudre des problèmes d'optimisation *NP-difficile*. Sur ces problèmes, en admettant que $P \neq NP$, il n'existe pas d'algorithme en temps polynomial et donc une résolution complète entraîne un temps de calcul exponentiel dans le pire des cas. Contrairement à ces approches, la recherche locale a pour objectif de trouver une solution en un temps fixé (souvent polynomial). Elle est basée sur deux concepts majeurs : (i) l'intensification qui consiste à fouiller une zone réduite de l'espace de recherche pour en extraire éventuellement une solution et (ii) la diversification qui permet à la recherche de se déplacer dans l'espace plus largement. Ce sont ces notions qui définissent le comportement des méthodes à base de recherche locale vis-à-vis de l'espace de recherche.

2.1.1 Définitions et notions de bases

Les méthodes de recherche locale ou métaheuristiques à base de voisinages s'appuient toutes sur un même principe. À partir d'une solution initiale (souvent générée de manière aléatoire), la recherche locale consiste à se déplacer d'interprétation en interprétation jusqu'à l'obtention d'un modèle. Le paysage ainsi décrit est un sous-ensemble de l'ensemble des interprétations et les modèles sont les minima (ou maxima) d'une fonction d'évaluation. L'objectif de ces méthodes est d'exhiber un modèle et donc d'atteindre un point dont l'altitude est minimale. Il est courant de parler d'altitude et de paysage « montagneux » lorsque l'on décrit le comportement d'une méthode de recherche locale. En effet, s'il était possible de représenter l'espace complet des interprétations en fonction du nombre de contraintes falsifiées, ce paysage ressemblerait certainement à un paysage alpin. Par conséquent, l'objectif de ces méthodes est d'atteindre le niveau de la mer, c'est-à-dire une altitude de zéro signifiant qu'aucune contrainte n'est falsifiée et donc qu'un modèle a été trouvé.

Les méthodes de recherche locale utilisent principalement deux fonctions : une fonction de voisinage et une fonction d'évaluation. Une fonction d'amélioration, basée sur la fonction d'évaluation, est également souvent utilisée.

Le voisinage représente des affectations autour de la configuration courante, accessibles en modifiant certains attributs (valeurs des variables) et est très souvent relatif au problème posé. Le voisinage se définit généralement comme suit :

Définition 2.1 (voisinage direct)**.** *Soit $\mathfrak{S}$ un espace de recherche, un voisinage est une fonction :*

$$\begin{array}{rccc} \mathcal{N} : & \mathfrak{S} & \longrightarrow & 2^{\mathfrak{S}} \\ & \mathcal{I} & \longmapsto & \mathfrak{I} \end{array}$$

qui associe à chaque élément $\mathcal{I}$ un ensemble de voisins $\mathcal{N}(\mathcal{I}) \subseteq 2^{\mathfrak{S}}$. $\mathcal{N}(\mathcal{I})$ est nommé voisinage de $\mathcal{I}$.

À partir d'une configuration courante, le choix du voisin à la prochaine étape se fait par la fonction d'évaluation. La fonction d'évaluation $eval$ permet d'estimer la qualité d'une interprétation $\mathcal{I}$ de l'ensemble des interprétations $\mathfrak{S}$. Dans le contexte des problèmes de satisfaction de contraintes cette fonction est très souvent en relation avec le nombre de contraintes falsifiées par l'interprétation $\mathcal{I}$.

Définition 2.2 (fonction d'évaluation)**.** *Soient $\mathcal{F}$ une formule et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{F}$, la fonction d'évaluation est définie par :*

$$\begin{array}{rccl} eval: & \mathfrak{F} \times \mathfrak{S} & \longrightarrow & \mathbb{N} \\ & (\mathcal{F}, \mathcal{I}) & \longmapsto & |\{\alpha \in \mathcal{F} \textit{ telle que } \mathcal{I} \not\models \alpha\}| \end{array}$$

Remarque 2.1. *Une valeur de zéro, pour cette fonction d'évaluation, indique qu'aucune contrainte n'est falsifiée et donc que l'interprétation $\mathcal{I}$ considérée est un modèle de la formule.*

Exemple 2.1. *Afin d'illustrer ces deux fonctions, considérons la figure 2.1. Sur cette figure, nous distinguons le chemin d'une recherche locale où la fonction d'évaluation des interprétations complètes permet de sélectionner un voisin à chaque pas.*

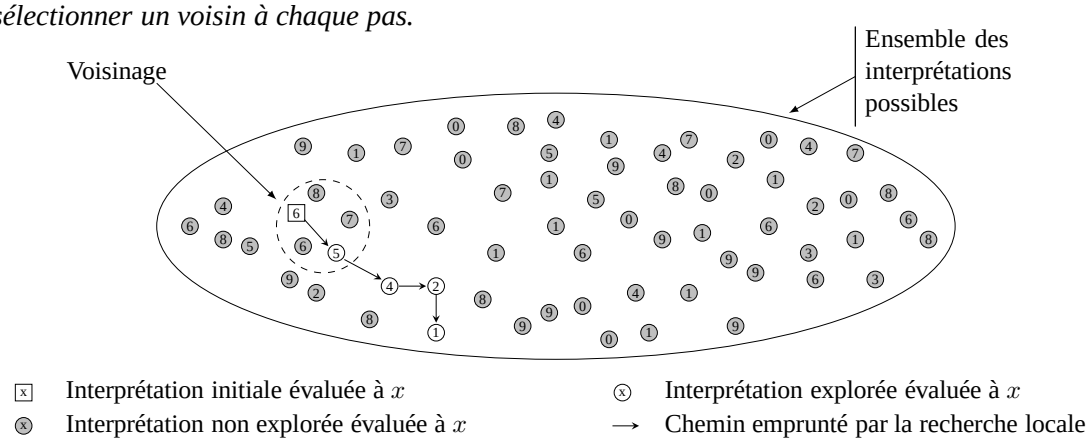


FIGURE 2.1 – Chemin de recherche locale.

La fonction $diff$ permet de connaître la valeur, *a posteriori*, de la fonction d'évaluation dans le cas où l'interprétation $\mathcal{I}$ se déplace vers une interprétation $\mathcal{I}' \in \mathcal{N}(\mathcal{I})$. Pour cela, il suffit de considérer la différence entre le nombre de contraintes falsifiées par $\mathcal{I}$ et celui de $\mathcal{I}'$.

Définition 2.3 (amélioration)**.** *Soient $\mathcal{F}$ une formule et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{F}$, la fonction permettant de calculer l'amélioration obtenue, dans le cas où le prochain mouvement conduit à l'interprétation voisine $\mathcal{I}'$, est définie de la manière suivante :*

$$\begin{array}{rccl} diff: & \mathfrak{F} \times \mathfrak{S} \times \mathfrak{S} & \longrightarrow & \mathbb{N} \\ & (\mathcal{F}, \mathcal{I}, \mathcal{I}') & \longmapsto & eval(\mathcal{F}, \mathcal{I}) - eval(\mathcal{F}, \mathcal{I}') \end{array}$$

Exemple 2.2. *La figure 2.2 illustre le résultat obtenu par la fonction $diff$ (flèche annotée) sur un ensemble d'interprétations (représentées avec leur évaluation en gris) vis-à-vis d'une interprétation initiale (représentée avec son évaluation en blanc).*

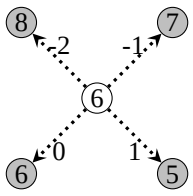


FIGURE 2.2 – Amélioration pouvant être obtenue en se déplaçant d'une interprétation à une autre.

Remarque 2.2. *Il est à noter que l'amélioration définie entre deux interprétation $\mathcal{I}$ et $\mathcal{I}'$ peut avoir une valeur $diff(\mathcal{F}, \mathcal{I}, \mathcal{I}')$ négative, positive ou nulle, ce qui indique que le déplacement de $\mathcal{I}$ vers $\mathcal{I}'$ respectivement dégrade, améliore ou ne modifie pas la valeur de la fonction d'évaluation.*

Ces fonctions, modulo quelques modifications, constituent les éléments de base des algorithmes de types « descente ». Ces algorithmes sont basés sur le principe que, tant qu'il existe une interprétation au voisinage de l'interprétation courante améliorant la fonction d'évaluation (c'est-à-dire $\exists \mathcal{I}' \in \mathcal{N}(\mathcal{I})$ telle que $diff(\mathcal{F}, \mathcal{I}, \mathcal{I}') > 0$), il faut se déplacer vers cette interprétation voisine. Ce déplacement est appelé réparation de l'interprétation courante.

Appliqué seul, le principe de descente conduit souvent à des minima. Un minimum est une interprétation qui n'est pas un modèle et pour laquelle il n'existe pas dans son voisinage d'interprétation permettant d'améliorer la valeur de la fonction d'évaluation, c'est-à-dire permettant de diminuer le nombre de contraintes falsifiées.

Définition 2.4 (minimum)**.** *Soient $\mathcal{F}$ une formule et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{F}$. On dit que $\mathcal{I}$ est un minimum de $\mathcal{F}$ par rapport à la fonction d'évaluation $eval$ si aucun déplacement vers une interprétation voisine de $\mathcal{I}$ ne permet d'améliorer $eval$. Formellement :*

$$\mathcal{I} \text{ est un minimum de } \mathcal{F} \text{ par rapport à } eval$$
$$\Leftrightarrow$$
$$\forall \mathcal{I}' \in \mathcal{N}(\mathcal{I}), eval(\mathcal{F}, \mathcal{I}') \geq eval(\mathcal{F}, \mathcal{I})$$

Nous distinguons deux types de minima : les *minima globaux* sont des interprétations telles qu'il n'existe aucune interprétation de la formule ayant une meilleure valeur de la fonction d'évaluation. Pour une formule satisfiable et la fonction d'évaluation qui consiste à considérer le nombre de contraintes falsifiées, la valeur des minima globaux est de 0. À l'inverse, les *minima locaux* sont des interprétations telles qu'il n'existe pas dans leur voisinage une interprétation améliorant la valeur de la fonction d'évaluation mais telles qu'il existe dans l'ensemble des interprétations au moins une interprétation dont la valeur de la fonction d'évaluation est meilleure (voir Figure 2.3).

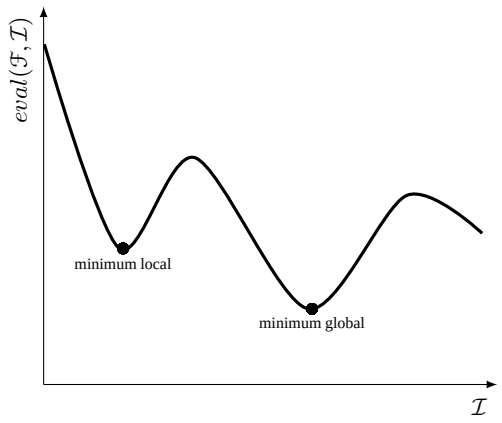


FIGURE 2.3 – Minimum local et global.

De manière générale, que ce minimum soit local ou global, il est important de pouvoir s'en échapper. Pour cela, il est nécessaire de permettre à l'opérateur de recherche locale d'effectuer des mouvements pour lesquels la nouvelle solution retenue sera de qualité moindre que la précédente. Ce principe, détaillé

dans la section 2.2, est appelé principe d'échappement et doit permettre de s'éloigner suffisamment de l'interprétation courante afin d'éviter de la revisiter une nouvelle fois. Il existe différentes stratégies pour s'échapper d'un minimum local (redémarrage, aléatoire, retour arrière, tabou, etc.) et leur tâche consiste à trouver le meilleur moyen de sortir ou d'éviter les minima locaux afin de poursuivre la recherche. C'est souvent ce principe qui différencie les multiples algorithmes de recherche locale. L'algorithme 2.1 décrit succinctement le calcul effectué par une méthode de recherche locale.

Algorithme 2.1 : Recherche Locale

Données : $\mathcal{F}$ une formule et $maxReparations$, le nombre maximum de réparations autorisées
Résultat : `vrai` si la formule $\mathcal{F}$ est satisfiable, `faux` s'il est impossible de conclure

1 **Début**
2 $\quad \mathcal{I} \longleftarrow$ une interprétation complète générée aléatoirement;
3 $\quad nbReparation = 0$;
4 $\quad$ **tant que** $(nbReparation < maxReparations)$ *et* $(eval(\mathcal{F}, \mathcal{I}) > 0)$ **faire**
5 $\quad\quad$ **si** $\exists \mathcal{I}' \in \mathcal{N}(\mathcal{I})$ *telle que* $diff(\mathcal{F}, \mathcal{I}, \mathcal{I}') > 0$ **alors** $\mathcal{I} \longleftarrow \mathcal{I}'$;
6 $\quad\quad$ **sinon** $\mathcal{I} \longleftarrow$ interprétation choisie suivant un critère d'échappement;
7 $\quad\quad nbReparation \longleftarrow nbReparation + 1$;
8 $\quad$ **si** $(eval(\mathcal{F}, \mathcal{I}) = 0)$ **alors retourner** `vrai` ;
9 $\quad$ **retourner** `faux` ;
10 **Fin**

Afin de concevoir un algorithme de recherche locale efficace, il est nécessaire de mettre au point des structures de données évoluées permettant le développement d'algorithmes incrémentaux puissants. Ces structures dépendent bien entendu du problème à traiter et sont discutées lors de la présentation de l'utilisation de la recherche locale dans le cadre spécifique des problèmes SAT (voir 2.1.4) et CSP (voir 2.1.5).

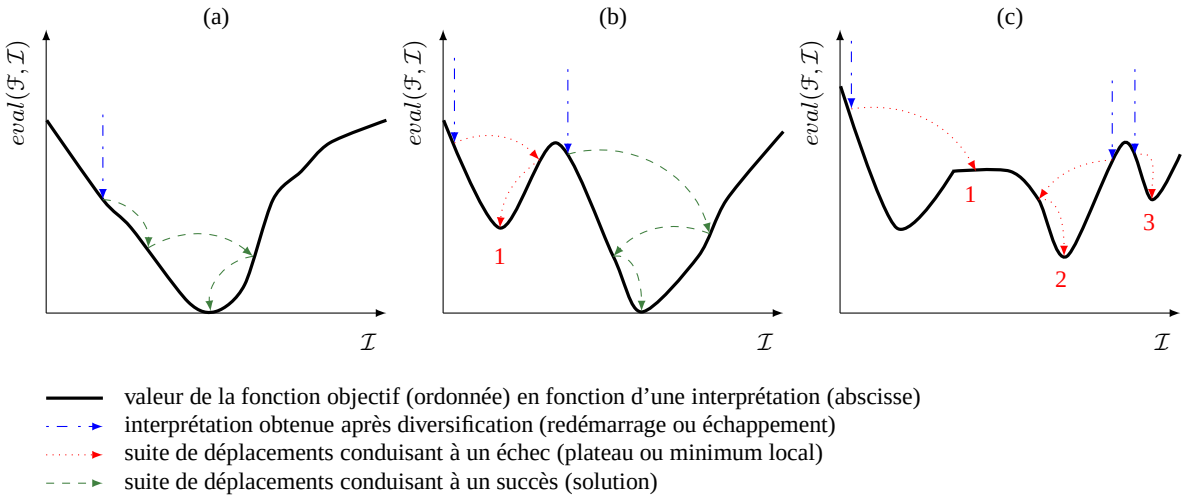


FIGURE 2.4 – Paysage exploré par la recherche locale.

Les méthodes de type recherche locale sont amenées à faire face à différentes situations. La Figure 2.4 détaille une étude des trois cas de figure pouvant être rencontrés par une méthode de type recherche

locale :

a. Sur l'image de gauche nous avons le cas « espéré ». Étant donné une interprétation complète générée aléatoirement, la méthode arrive à atteindre un minimum global lequel est aussi une solution du problème considéré.

b. Le schéma central correspond au cas où la recherche locale atteint un minimum local (situation 1 sur le schéma (b)). Puisqu'un seul déplacement ne permet plus d'améliorer la valeur de la fonction d'évaluation, une étape de diversification est opérée (redémarrage ou application d'un critère d'échappement). Une fois soustrait du minimum local la méthode arrive au bout d'un certain nombre d'étapes à trouver une solution.

c. L'image de droite schématise le cas où l'instance considérée est insatisfiable ou le cas où la recherche locale a échoué à trouver un modèle. Comme on peut le voir, la valeur de la fonction d'évaluation est toujours supérieure à zéro. En d'autres mots, quelle que soit l'interprétation considérée, le nombre de contraintes falsifiées est toujours différent de zéro. Cette image dépeint aussi le cas particulier où un ensemble d'interprétations voisines forme un *plateau* (situation 1 sur le schéma (c)). Nous appelons plateaux un ensemble d'interprétations voisines ayant la même valeur pour la fonction objectif. Comme nous pouvons le voir, sur ce dernier cas de figure, la recherche locale atteint toujours soit un minimum soit un plateau (situation 2 et 3 sur le schéma (c)).

Dans le cadre des problèmes SAT et CSP l'algorithme de recherche locale précédemment décrit est légèrement modifié. Ces modifications caractérisent l'approche GSAT présentée dans la partie suivante.

2.1.2 L'architecture GSAT

L'algorithme GSAT est certainement l'algorithme le plus connu des méthodes de recherche locale pour SAT. Cet algorithme est introduit initialement par Mitchell *et al.* (1992), mais ce n'est que lors du « *Second Challenge on Satisfiability Testing* » de DIMACS, que des variantes extrêmement efficaces de cet algorithme ont été proposées (Selman *et al.* 1993).

Il existe peu de différences entre GSAT et la méthode de recherche locale « classique » présentée précédemment. La fonction d'évaluation pour cette approche reste la plus naturelle qui soit, c'est-à-dire le nombre de contraintes falsifiées par l'interprétation courante (« min-conflict » (Minton *et al.* 1990)). Le voisinage utilisé est un « voisinage direct », c'est-à-dire que les interprétation voisines sont toutes les interprétations qui diffèrent de l'interprétation courante que par la valeur d'une variable (c'est-à-dire se trouvant à une distance de Hamming de 1).

Définition 2.5 (Voisinage de GSAT)**.** *Soient $\mathcal{F}$ une formule et $\mathcal{I}$ une interprétation complète construite sur l'ensemble des variables de $\mathcal{F}$. On a :*

$$\mathcal{N}_{\text{GSAT}}(\mathcal{I}) = \{\mathcal{J} \textit{ telle que } \mathcal{J} \textit{ est une interprétation de } \mathcal{F} \textit{ et } \delta_{\mathfrak{h}}(\mathcal{I}, \mathcal{J}) = 1\}.$$

La description de GSAT peut se faire de manière très succincte tant cet algorithme est simple. L'algorithme commence par générer aléatoirement une interprétation complète des variables de la formule. Une fois celle-ci établie, l'algorithme effectue un certain nombre de réparations jusqu'à l'obtention d'un modèle. Ces réparations consistent à modifier la valeur d'une variable permettant d'obtenir le meilleur gain (s'il y en a plusieurs, l'algorithme en choisit une de manière aléatoire). Si, après un certain nombre d'étapes $maxReparations$, la recherche locale n'a toujours pas trouvé de solution, la procédure recommence avec une nouvelle interprétation initiale. Ce processus peut se répéter un nombre maximal $maxEssais$ de fois fixé au démarrage avant de retourner que la recherche a échoué.

GSAT ne présente pas, comme pour l'algorithme de base, deux phases bien distinctes : descente et échappement. En effet, cette approche autorise des dégradations de la fonction d'évaluation au cours de la recherche, ce qui peut être considéré comme un moyen de s'échapper d'un minimum local. Par ailleurs, il existe une autre phase d'échappement dans cette méthode qui consiste à redémarrer la recherche avec une nouvelle configuration.

Malgré cela, il reste que cette approche conduit le plus souvent à un minimum local. C'est pourquoi de nombreuses variantes de cet algorithme ont vu le jour. Ces adaptations sont basées sur un schéma légèrement différent de celui de GSAT. Ce schéma, présenté dans l'algorithme 2.2, permet d'intégrer facilement différentes stratégies d'échappements (lesquelles sont présentées dans la section 2.2).

Algorithme 2.2 : GSAT

Données : $\mathcal{F}$ une formule, deux entiers $maxReparations$ et $maxEssais$
Résultat : `vrai` si la formule $\mathcal{F}$ est satisfiable, `faux` s'il est impossible de conclure

```
1 Début
2   pour i de 0 à maxEssais faire
3     I ⟵ une interprétation complète générée aléatoirement;
4     pour j de 0 à maxReparations faire
5       si (eval(F, I) = 0) alors retourner vrai ;
6       si (∃I' ∈ N_GSAT(I)) telle que diff(F, I, I') > 0 alors I ⟵ I';
7       sinon I ⟵ interprétation choisie suivant un critère d'échappement;
8   retourner (eval(F, I) = 0);
9 Fin
```

Une des clefs du succès de cette approche tient au fait que seul un sous-ensemble des interprétations voisines est considéré pendant la phase de descente. En effet, puisque cette phase consiste à satisfaire les contraintes non satisfaites par l'interprétation courante, il est possible de limiter le voisinage direct en ne considérant que les variables appartenant aux contraintes falsifiées par l'interprétation courante. Il n'est pas rare, dans le cadre de la résolution pratique de problème de satisfaction de contraintes, d'être confronté à des problèmes de très grandes tailles (plus d'un million de variables) et donc à un voisinage conséquent. Par conséquent, considérer uniquement cet ensemble d'interprétations voisines lors de la phase de descente permet alors un gain substantiel.

Une autre amélioration qui est l'une des plus importantes pour ces méthodes, consiste à intégrer la stratégie de « marche aléatoire » pendant la résolution, produisant ainsi l'algorithme WSAT.

2.1.3 L'architecture WSAT

WSAT ou « *WalkSAT* » (Selman *et al.* 1994) est l'héritier de GSAT. Cette approche introduit le concept de marche aléatoire pendant la résolution. La marche aléatoire permet d'accélérer le processus de recherche tout en permettant de s'échapper de certains minima locaux. Dans cette stratégie, le choix de la variable à réviser est restreint aux variables impliquées dans une contrainte falsifiée choisie aléatoirement. Ainsi, seul un sous-ensemble du voisinage direct est évalué, permettant d'augmenter significativement la fréquence à laquelle les mouvements dans l'espace de recherche sont effectués. Ce dernier procédé permet aussi d'introduire une phase de diversification supplémentaire par le biais du choix de la dite contrainte falsifiée.

Définition 2.6 (Voisinage de WSAT)**.** *Soient $\mathcal{F}$ une formule, $\mathcal{I}$ une interprétation complète construite sur l'ensemble des variables de $\mathcal{F}$ et α une contrainte de $\mathcal{F}$ falsifiée par $\mathcal{I}$. On a :*

$$\mathcal{N}_{\text{WSAT}}(\mathcal{I}, \alpha) = \{\mathcal{J} \text{ tel que } \mathcal{J} \text{ est une interprétation de } \mathcal{F} \text{ et } \zeta_{\mathfrak{h}}(\mathcal{I}, \mathcal{J}) = \{x\} \text{ avec } x \in \alpha\}.$$

La fonction d'évaluation reste « *min-conflict* », à savoir que la variable réparée est celle qui permet de réduire au maximum le nombre de contraintes falsifiées. L'algorithme 2.3, comme pour GSAT, décrit un schéma d'algorithme reprenant les fondements de WSAT et permettant l'intégration de différents critères d'échappement.

Algorithme 2.3 : WSAT

Données : $\mathcal{F}$ une formule, deux entiers $maxReparations$ et $maxEssais$
Résultat : `vrai` si la formule $\mathcal{F}$ est satisfiable, `faux` s'il est impossible de conclure

```
1  Début
2      pour i de 0 à maxEssais faire
3          I ⟵ une interprétation complète générée aléatoirement;
4          pour j de 0 à maxReparations faire
5              si (eval(F, I) = 0) alors retourner vrai ;
6              α ⟵ une contrainte au hasard parmi les contraintes falisifiées par I;
7              si (∃I' ∈ N_WSAT(I, α) telle que diff(F, I, I') > 0) alors I ⟵ I';
8              sinon I ⟵ interprétation choisie suivant un critère d'échappement;
9      retourner (eval(F, I) = 0);
10 Fin
```

Dans la suite nous introduisons les différentes notions nécessaires (voisinage, fonction d'évaluation et calcul de l'amélioration) à l'utilisation des schémas algorithmiques présentés précédemment dans le cadre de la résolution pratique des problèmes SAT et CSP.

2.1.4 Application de la recherche locale au problème SAT

Afin de pouvoir réaliser les différents schémas de recherche locale présentés précédemment, il est indispensable d'en définir les briques élémentaires. Ces briques sont les fonctions d'évaluation, de voisinage et d'amélioration.

Comme précisé auparavant, la fonction d'évaluation est définie par rapport au nombre de contraintes falsifiées par l'interprétation courante. Ce qui s'exprime formellement par :

Définition 2.7 (fonction évaluation)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation complète construite sur les variables de Σ, la fonction d'évaluation est définie par :*

$$\begin{array}{rccl} eval: & \boldsymbol{\Sigma} \times \mathfrak{S} & \longrightarrow & \mathbb{N} \\ & (\Sigma, \mathcal{I}) & \longmapsto & |\{\alpha \in \alpha \text{ telle que } \mathcal{I} \not\models \alpha\}| \end{array}$$

Exemple 2.3. *Soient $\Sigma = \{(a \vee b \vee c), (\neg a \vee b), (\neg a \vee b \vee d), (b \vee \neg c), (\neg a \vee \neg c)\}$ une formule et $\mathcal{I} = \{a, \neg b, \neg c, \neg d\}$ une interprétation complète construite sur les variables de Σ, alors $eval(\Sigma, \mathcal{I}) = 2$.*

Dans le cadre de SAT, les interprétations voisines sont toutes les interprétations pouvant être obtenues en inversant la valeur de vérité d'une variable. Cette opération, appelée flip, est définie comme suit.

Définition 2.8 (flip)**.** *Soit $\mathcal{I}$ une interprétation complète construite sur un ensemble de variables propositionnelles, le flip de $x \in \mathcal{I}$ est défini par :*

$$\begin{array}{rccc} flip: & \mathfrak{S} \times \mathcal{V} & \longrightarrow & \mathfrak{S} \\ & (\mathcal{I}, x) & \longmapsto & \{\mathcal{I} \setminus \{x\}\} \cup \{\tilde{x}\} \end{array}$$

Les différentes notions de voisinage se définissent à présent naturellement comme :

Définition 2.9 (voisinage direct)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation complète construite sur les variables de Σ et $\mathcal{V}$ un ensemble de variables tel que $\mathcal{V} \subseteq \mathcal{V}_\Sigma$. L'ensemble des interprétations voisines de $\mathcal{I}$ pouvant être atteintes à partir de $\mathcal{V}$ est défini comme suit :*

$$\begin{array}{rccc} \mathcal{N}: & \mathfrak{S} \times 2^{\mathcal{V}_\Sigma} & \longrightarrow & 2^{\mathfrak{S}} \\ & (\mathcal{I}, \mathcal{V}) & \longmapsto & \{\mathcal{I}' \in \mathfrak{S} \text{ tel que } \mathcal{I}' = flip(\mathcal{I}, x), \text{ avec } x \in \mathcal{V}\} \end{array}$$

Définition 2.10 (voisinage GSAT)**.** *Soient Σ une formule et $\mathcal{I}$ une interprétation complète construite sur l'ensemble des variables de Σ. $\mathcal{N}_{\text{GSAT}}(\mathcal{I}) = \mathcal{N}(\mathcal{I}, \mathcal{V}_g)$ avec $\mathcal{V}_g = \{x \in \alpha$ tel que $\alpha \in \Sigma$ et $\mathcal{I} \not\models \alpha\}$ l'ensemble des variables apparaissant dans les contraintes falsifiées par l'interprétation $\mathcal{I}$.*

Définition 2.11 (voisinage WSAT)**.** *Soient Σ une formule et $\mathcal{I}$ une interprétation complète construite sur les variables de Σ. $\mathcal{N}_{wsat} = \mathcal{N}(\mathcal{I}, \mathcal{V}_\alpha)$ avec $\mathcal{V}_\alpha = \{x \in \alpha\}$ tel que $\alpha \in \Sigma$ et $\mathcal{I} \not\models \alpha$.*

Exemple 2.4. *Considérons la formule Σ et l'interprétation $\mathcal{I}$ de l'exemple 2.3. L'ensemble des interprétations voisines de l'interprétation $\mathcal{I}$ pour les différentes notions de voisinages présentées sont :*

- *direct : $\mathcal{N}(\mathcal{I}, \mathcal{V}_\Sigma) = \{\{\neg a, \neg b, \neg c, \neg d\}, \{a, b, \neg c, \neg d\}, \{a, \neg b, c, \neg d\}, \{a, \neg b, \neg c, d\}\}$;*
- *GSAT : $\mathcal{N}(\mathcal{I}, \mathcal{V}_g) = \{\{\neg a, \neg b, \neg c, \neg d\}, \{a, b, \neg c, \neg d\}, \{a, \neg b, \neg c, d\}\}$;*
- *WSAT : $\mathcal{N}(\mathcal{I}, \mathcal{V}_\alpha) = \{\{\neg a, \neg b, \neg c, \neg d\}, \{a, b, \neg c, \neg d\}\}$ avec $\alpha = (\neg a \vee b)$ une clause de Σ falsifiée par $\mathcal{I}$.*

Les notions de voisinage GSAT et WSAT ont des propriétés intéressantes. Une première propriété permet d'établir que l'ensemble des interprétations considérées par la notion de voisinage WSAT est inclus dans l'ensemble des interprétations considérées par la notion de voisinage GSAT.

Propriété 2.1. *Soient Σ une formule et $\mathcal{I}$ une interprétation complète de Σ, alors $\forall \alpha \in \Sigma$ telle que $\mathcal{I} \not\models \alpha$, $\mathcal{N}(\mathcal{I}, \alpha) \subseteq \mathcal{N}(\mathcal{I}, \mathcal{V}_g)$.*

Preuve 2.1. *La preuve est triviale et vient du fait que $\forall \alpha \in \Sigma$ falsifiée par $\mathcal{I}$ nous avons $\alpha \subseteq \mathcal{V}_g$.*

Exemple 2.5. *Sur l'exemple 2.4 nous voyons clairement que cette propriété est vérifiée.*

Cette propriété, quoique triviale, explique en partie le gain de performance obtenu en faveur de l'architecture WSAT vis-à-vis de l'architecture GSAT. Deux autres propriétés importantes permettent d'assurer qu'au moins une clause falsifiée par une interprétation $\mathcal{I}$ sera satisfaite par une interprétation voisine $\mathcal{I}'$ obtenue à partir d'une des notions de voisinage GSAT ou WSAT.

Propriété 2.2. *Soient Σ une formule et $\mathcal{I}$ une interprétation complète de Σ, alors $\forall \mathcal{I}' \in \mathcal{N}(\mathcal{I}, \mathcal{V}_g)$ il existe $\alpha \in \Sigma$ tel que $\mathcal{I} \not\models \alpha$ et $\mathcal{I}' \models \alpha$.*

Preuve 2.2. *Considérons $\mathcal{I}' \in \mathcal{N}(\mathcal{I}, \mathcal{V}_g)$, par définition de la notion de voisinage GSAT nous avons que $\exists x \in \mathcal{I}$ tel que $\tilde{x} \in \mathcal{I}'$ et que $\exists \alpha \in \Sigma$ tel que $\tilde{x} \in \alpha$ et $\mathcal{I} \not\models \alpha$. ce qui implique que $\mathcal{I}' \models \alpha$.*

Exemple 2.6. *Considérons la formule Σ et l'interprétation $\mathcal{I} = \{\neg a, \neg b, \neg c, \neg d\}$ de l'exemple 2.4. Supposons que l'interprétation choisie comme interprétation voisine soit $\mathcal{I}' = \{a, \neg b, \neg c, \neg d\}$. Dans ce cas, nous remarquons que la clause $(\neg a \vee b \vee d)$ est falsifiée par $\mathcal{I}$ et satisfaite par $\mathcal{I}'$.*

Propriété 2.3. *Soient Σ une formule et $\mathcal{I}$ une interprétation complète de Σ, alors $\forall \mathcal{I}' \in \mathcal{N}(\mathcal{I}, \mathcal{V}_\alpha)$ il existe $\alpha \in \Sigma$ tel que $\mathcal{I} \not\models \alpha$ et $\mathcal{I}' \models \alpha$.*

Preuve 2.3. *Les propriétés 2.1 et 2.2 permettent de conclure à la validité de cette propriété.*

Nous verrons que, quoique triviales dans le cadre de SAT, ces deux dernières propriétés sont difficiles à obtenir dans le cadre de CSP.

La construction d'algorithmes de recherche locale efficaces nécessite la mise au point de structures de données évoluées, permettant de gérer efficacement l'évaluation des interprétations voisines. En effet, le temps de calcul de ces algorithmes est déterminé non seulement par le cadre de recherche, mais aussi par la complexité de chaque itération. L'utilisation de structures de données adéquates permet alors de procéder efficacement aux déplacements (flips) et au choix de la prochaine variable à flipper. La plupart des solveurs SAT de type recherche locale introduisent deux vecteurs d'entiers nommés $breakcount$ et $makecount$ permettant respectivement de donner le nombre de clauses que falsifie et satisfait le flip d'une variable. De manière à obtenir de bonnes performances il est nécessaire que ces deux tableaux soient maintenus à jour efficacement tout au long de la recherche. En vue de réaliser cela, Fukunaga (2004) propose une approche basée sur la notion de « *watched literals* » (présentée dans la section 3.1.4.2) afin d'augmenter significativement leur mise à jour. Grâce à ces structures la fonction d'amélioration se calcule facilement de la manière suivante :

Définition 2.12 (amélioration)**.** *Soient Σ une formule et $\mathcal{I}$ une interprétation complète construite sur les variables de Σ, la fonction d'amélioration obtenue, dans le cas où le prochain mouvement conduit à flipper la variable x, est définie de la manière suivante :*

$$\begin{array}{rccl} diff: & \boldsymbol{\Sigma} \times \mathfrak{S} \times x & \longrightarrow & \mathbb{N} \\ & (\Sigma, \mathcal{I}, x) & \longmapsto & makecount[x] - breakcount[x] \end{array}$$

La recherche locale dans le cadre de SAT a été largement étudiée et plusieurs solveurs ont été proposés (WALKSAT43 (Kautz et Selman), UBCSAT (Tompkins), etc.). Dans la suite, nous présentons les éléments de base permettant l'utilisation de la recherche locale dans le cadre du problème CSP.

2.1.5 Application de la recherche locale au problème CSP

Dans cette partie, tout comme pour SAT, nous définissons les briques nécessaires à l'élaboration d'un solveur de type recherche locale pour la résolution pratique du problème CSP.

La fonction d'évaluation reste la plus naturelle qui soit, c'est-à-dire *min-conflict* [11].

Définition 2.13 (fonction d'évaluation)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{X}$, la fonction d'évaluation est définie par :*

$$\begin{array}{rccl} eval: & \mathfrak{P} \times \mathfrak{S} & \longrightarrow & \mathbb{N} \\ & (\mathcal{P}, \mathcal{I}) & \longmapsto & |\{\alpha \in \mathcal{C} \text{ tel que } \mathcal{I} \not\models \alpha\}| \end{array}$$

Exemple 2.7. *Considérons le réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ dépeint dans la figure 2.5 et l'interprétation complète $\mathcal{I} = \{(X_1, 1), (X_2, 5), (X_3, 1), (X_4, 6), (X_5, 2)\}$, nous avons $eval(\mathcal{P}, \mathcal{I}) = 2$.*

11. Une autre notion de voisinage introduite par Nonobe et Ibaraki (1998) consiste à associer pour toutes les variables du problème une variable booléenne pour chacune de ces valeurs. Ces variables booléennes permettent de définir si la valeur est choisie ou non pour la variable. Dans cette encodage, le problème est redéfini à l'aide de contraintes de cardinalité et d'inégalité linéaire et quadratique.

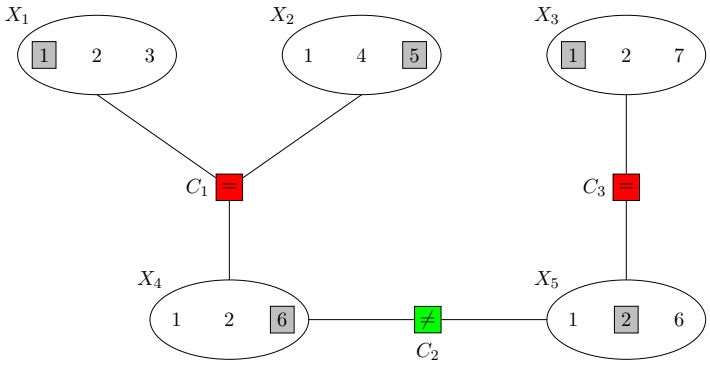


FIGURE 2.5 – Interprétation complète définie sur un réseau de contraintes tel que les valeurs encadrées représentent les valeurs choisies par l'interprétation et les contraintes coloriées en rouge (respectivement vert) représentent les contraintes falsifiées (respectivement satisfaites).

Pour CSP, les interprétations voisines sont toutes les interprétations pouvant être atteintes si la valeur d'une variable est modifiée par une autre valeur de son domaine. Cette opération, appelé $switch$, est définie comme suit :

Définition 2.14 (switch)**.** *Soit $\mathcal{I}$ une interprétation complète construite sur un ensemble de variables discrètes $\mathcal{X}$. Considérons le couple $(X, u) \in \mathcal{I}$ tel que $X \in \mathcal{X}$ et $u \in X$. Le $switch$ de la valeur u de la variable $dom(X)$ en $v \in dom(X)$ est défini de la manière suivante :*

$$\begin{array}{rccl} switch: & \mathfrak{S} \times \mathcal{X} \times dom(X) & \longrightarrow & \mathfrak{S} \\ & (\mathcal{I}, X, v) & \longmapsto & \{\mathcal{I} \setminus \{(X,u)\}\} \cup \{\{(X,v)\}\} \end{array}$$

Exemple 2.8. *Soit $\mathcal{I} = \{(X_1,1),(X_2,5),(X_3,1),(X_4,6),(X_5,2)\}$, appliquer $switch(\mathcal{I}, X_1, 3)$ conduit à l'interprétation $\mathcal{I} = \{(X_1,3),(X_2,5),(X_3,1),(X_4,6),(X_5,2)\}$.*

Les différentes notions de voisinages se définissent à présent naturellement.

Définition 2.15 (voisinage direct)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{X}$, le voisinage direct de $\mathcal{I}$ est défini comme suit :*

$$\begin{array}{rccl} \mathcal{N}: & \mathfrak{S} \times 2^{\mathcal{X}} & \longrightarrow & 2^{\mathfrak{S}} \\ & (\mathcal{I}, \mathcal{V}_{\mathcal{X}}) & \longmapsto & \{\mathcal{I}' \in \mathfrak{S} \text{ tel que } \mathcal{I}' = switch(\mathcal{I}, X, v) \text{ et } \mathcal{I} \neq \mathcal{I}'\} \end{array}$$

avec $X \in \mathcal{V}_{\mathcal{X}}$ et $v \in dom(X)$.

Définition 2.16 (voisinage GSAT)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{X}$. $\mathcal{N}_{\text{GSAT}}(\mathcal{I}) = \mathcal{N}(\mathcal{I}, \mathcal{V}_g)$ avec $\mathcal{V}_g = \{X \in C$ tel que $C \in \mathcal{C}$ et $\mathcal{I} \not\models C\}$ l'ensemble des variables apparaissant dans les contraintes falsifiées par l'interprétation $\mathcal{I}$.*

Définition 2.17 (voisinage WSAT)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes et $\mathcal{I}$ une interprétation complète construite sur les variables de $\mathcal{X}$. $\mathcal{N}_{\text{WSAT}} = \mathcal{N}(\mathcal{I}, C)$ telle que $C \in \mathcal{C}$ est une contrainte falsifiée par $\mathcal{I}$.*

Exemple 2.9. *Considérons le réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ et l'interprétation $\mathcal{I}$ de l'exemple 2.7. Un ensemble d'interprétations voisines de $\mathcal{I}$ pour la notion de voisinages* WSAT [12] *pour la contrainte C_3*

12. Étant donné le nombre important d'interprétations voisines pour les notions de voisinages direct et GSAT (10 interprétations par notion), nous ne donnons pas d'exemple. Nous soulignons tout de même que ces deux ensembles sont identiques. En effet, toutes les variables du réseau apparaissent dans des contraintes falsifiées par l'interprétation courante.

est :

$$\mathcal{N}(\mathcal{I}, C_3) = \{\{(X_1,1),(X_2,5),(X_3,2),(X_4,6),(X_5,2)\},\{(X_1,1),(X_2,5),(X_3,7),(X_4,6),(X_5,2)\},\\ \{(X_1,1),(X_2,5),(X_3,1),(X_4,6),(X_5,1)\},\{(X_1,1),(X_2,5),(X_3,1),(X_4,6),(X_5,6)\}\}$$

À présent nous observons si les propriétés 2.1, 2.2 et 2.3 énoncées précédemment dans le cadre de SAT sont vérifiées dans le cadre CSP. En ce qui concerne la propriété 2.1, elle est toujours vérifiée pour la notion de voisinage définie.

Propriété 2.4. *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes et $\mathcal{I}$ une interprétation complète de $\mathcal{X}$, alors $\forall C \in \mathcal{C}$ telle que $\mathcal{I} \not\models C$, $\mathcal{N}(\mathcal{I}, C) \subseteq \mathcal{N}(\mathcal{I}, \mathcal{V}_g)$.*

Preuve 2.4. *Voir preuve 2.1.*

En ce qui concerne les propriétés 2.2 et 2.3, elles ne sont pas vérifiées dans le cadre CSP. En effet, considérons le réseau de contraintes et l'interprétation complète $\mathcal{I}$ de l'exemple 2.7. Supposons que l'interprétation voisine sélectionnée est $\mathcal{I}' = \{(X_1,1),(X_2,5),(X_3,7),(X_4,6),(X_5,2)\}$, dans ce cas l'ensemble des contraintes falsifiées par $\mathcal{I}$ est identique à celui falsifié par $\mathcal{I}'$.

Une autre particularité importante, dans le cadre CSP, est qu'il est impossible de juger si intervertir une valeur d'une variable permet de se rapprocher d'une interprétation permettant de satisfaire une nouvelle contrainte. En effet considérons le réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$, l'interprétation complète $\mathcal{I}$ de l'exemple 2.7 et l'interprétation $\mathcal{I}' = \{(X_1,1),(X_2,1),(X_3,1),(X_4,6),(X_5,2)\}$ voisine de $\mathcal{I}$. La fonction d'évaluation $eval$ définie précédemment évalue de manière identique $\mathcal{I}$ et $\mathcal{I}'$. Néanmoins, puisque cette interprétation permet de se rapprocher d'une solution permettant de satisfaire la contrainte C_1 (sans falsifier d'autres contraintes), il semble logique que $eval(\mathcal{P}, \mathcal{I})$ soit supérieure à $eval(\mathcal{P}, \mathcal{I}')$. Ce caractère particulier de la fonction d'évaluation rend plus difficile l'application de la recherche locale dans le cadre du problème CSP. Afin d'y remédier, une approche proposée par Galinier et Hao (2004) consiste à redéfinir la fonction d'évaluation afin de considérer certaines particularités associées à la sémantique des contraintes (contraintes de cardinalité, de distance, etc.). Dans le chapitre 8, nous montrons expérimentalement que ne pas considérer ce phénomène peut conduire à une approche de recherche locale inefficace sur les problèmes de grande arité.

Comme pour le problème SAT, afin de définir un algorithme de recherche locale efficace, il est primordial que la fonction permettant de calculer l'amélioration soit efficacement implémentée. Pour effectuer cela, la plupart des solveurs de recherche locale pour CSP introduisent une structure proposée par Galinier et Hao (1997). Ces structures de données et les algorithmes permettant de les maintenir à jour tout au long de la recherche sont présentés et perfectionnés dans le chapitre 8. Cette amélioration consiste à ajouter une variable booléenne pour chaque valeur de chaque variable du problème afin de conserver une trace des calculs antérieurs. Ces informations ainsi conservées permettent d'augmenter significativement la rapidité avec laquelle les structures de données sont maintenues.

L'ensemble des définitions présentées précédemment permettent d'établir le squelette d'un solveur de recherche locale. Néanmoins, pour être efficace, ce solveur doit être capable de parcourir l'espace de recherche « intelligemment ». L'idée est donc de ne pas visiter toutes les configurations, mais de se doter d'heuristiques pour choisir les zones d'exploration. Ces méthodes présentées dans la section suivante correspondent à des schémas généraux appelés métaheuristiques à base de voisinage ou stratégies d'échappement.

2.2 Stratégies d'échappement

Une des difficultés majeures liées à l'utilisation de la recherche locale consiste à réussir une exploration de l'espace de recherche sans stagner dans des minima locaux. En effet, la recherche aboutissant le plus souvent à un minimum local, les techniques d'échappement jouent un rôle primordial pour l'efficacité de ce type d'approches. Néanmoins, au vu du nombre réduit d'interprétations pouvant être examinées, déterminer l'interprétation qui, parmi un ensemble d'interprétations, est celle qui est la plus proche de l'objectif (c'est-à-dire celle qui permettra d'atteindre un modèle avec le plus petit nombre de déplacements), est une tâche délicate. Par conséquent, réaliser le meilleur déplacement, c'est-à-dire choisir le « meilleur » voisin, constitue la motivation principale des méthodes de recherche locale. La littérature sur le sujet foisonne, nous présentons dans la suite une liste non exhaustive de différents critères d'échappement parmi les plus connus et utilisés à la fois dans le cadre du problème SAT et du problème CSP.

2.2.1 Mouvements aléatoires

Dans le but d'échapper aux minima locaux, il est judicieux de ne pas effectuer le déplacement optimal à chaque pas. Cependant, pour trouver un modèle, le parcours de l'espace d'interprétations doit être pertinent, et la méthode de recherche locale doit appliquer des coups optimaux un certain nombre de fois. Partant de ce principe Selman et Kautz (1993) pour SAT et Minton *et al.* (1992) pour CSP proposent une approche, nommée RANDOMWALK, qui consiste à introduire des mouvements aléatoires dans le processus d'exploration. Pour cela, une probabilité p est introduite et appliquée de la manière suivante :

- avec une probabilité p, choisir aléatoirement une variable apparaissant dans une contrainte falsifiée et modifier sa valeur ;
- avec une probabilité $1 - p$, effectuer le coup optimal.

Il est possible d'imaginer une variante à cet algorithme où on ne restreint pas le choix de la variable aux contraintes falsifiées mais à l'ensemble des variables ou encore aux variables d'une contrainte falsifiée particulière.

2.2.2 La méthode Tabou

Cette approche proposée par Mazure *et al.* (1997) dans le cadre de SAT et Galinier et Hao (1997) pour CSP est une adaptation de la méthode tabou (Glover 1989; 1990). Dans cette méthode, le tabou, c'est-à-dire l'interdiction de choisir telle ou telle configuration, ne porte pas sur les interprétations mais sur les variables. Pour cela, les dernières variables qui ont été révisées sont conservées dans une file d'attente appelée « tabou ». Ensuite, lorsqu'une nouvelle variable est choisie, les variables appartenant à cette liste ne sont pas autorisées à être corrigées. Cette technique s'avère relativement efficace pour échapper aux minima, cependant ses performances dépendent de la taille de la liste tabou, dont la valeur optimale varie en fonction de l'instance à traiter. Une variante de cette approche consiste à utiliser un critère « d'aspiration ». Ce critère signifie que, sous certaines conditions, une variable appartenant à la liste tabou peut être modifiée (exemple : lorsque cette modification permet d'atteindre une valeur de la fonction d'évaluation jamais atteinte jusqu'alors).

2.2.3 Pondération de Contraintes

Dans cette méthode initiée par Morris (1993) et nommée « BREAKOUT », l'échappement des minima locaux est effectué à travers une redéfinition de la fonction objectif. En effet, plutôt que de compter

le nombre de contraintes falsifiées par chaque interprétation parcourue, un compteur initialisé à 1 est associé à chaque contrainte de la formule. Par rapport à ce nouvel élément, la fonction à minimiser est la somme des poids de chaque contrainte (Schuurmans *et al.* 2001, Eisenberg et Faltings 2003). Ainsi, en début de recherche, cette fonction est équivalente à celle classiquement définie. Cependant, à chaque fois qu'une contrainte est falsifiée par l'interprétation courante, son poids est incrémenté de 1. Avec cette modification dynamique de la fonction objectif, les minima locaux sont progressivement atténués jusqu'à disparaître, même en n'effectuant que des mouvements optimaux. La figure 2.6 montre schématiquement comment la pondération des contraintes falsifiées permet d'échapper aux minima locaux.

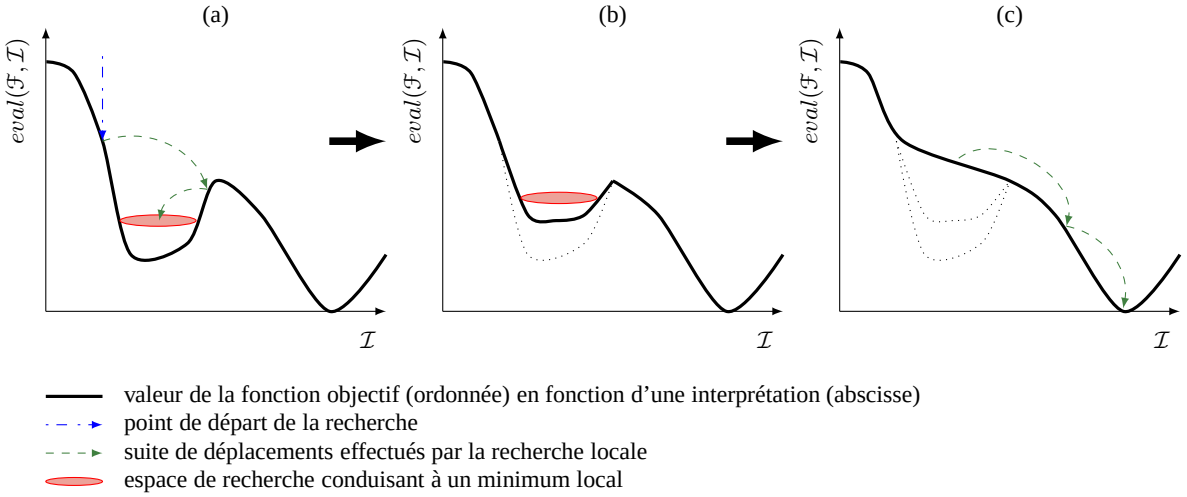


FIGURE 2.6 – Influence de la pondération des contraintes falsifiées sur le paysage de recherche.

Exemple 2.10. *Considérons l'espace de recherche arbitraire d'une formule quelconque $\mathcal{F}$. De plus, supposons que les premiers mouvements effectués par le processus de recherche locale conduisent à un minimum local (figure 2.6 courbe (a)). Dans ces conditions, ne pas diversifier la recherche conduit à considérer encore et encore le même minimum.*

Puisque la recherche locale reste engluée dans ce minimum, les contraintes qui sont souvent insatisfaites (c'est-à-dire les contraintes responsables de cette situation d'échec) voient leur poids devenir de plus en plus important. Pour être exact, leur poids est augmenté de 1 chaque fois qu'elles sont falsifiées. Cette augmentation implique un nivellement de l'espace de recherche (figure 2.6 courbe (b)).

Lorsqu'un certain nombre d'étapes a permis d'augmenter suffisamment le poids des contraintes responsables du minimum local, le solveur peut modifier la valeur d'une variable apparaissant dans l'une de ces contraintes (une variable apparaissant dans les contraintes ayant le poids le plus élevé). Ce processus guide alors la recherche dans une région de l'espace de recherche différente (figure 2.6 courbe (c)).

Bien que la formule $\mathcal{F}$ ne soit jamais modifiée durant tout le processus de recherche, nous pouvons voir sur la figure 2.6 qu'après un certain nombre d'étapes le paysage de recherche est modifié. Comme nous pouvons le voir sur cette figure, la recherche est guidée hors du minimum local lorsque l'augmentation de certaines contraintes a permis de « lisser » le minimum local.

Contrairement aux autres approches, l'augmentation du poids de certaines contraintes permet de pondérer les clauses appartenant aux parties difficiles du problème considéré. Ce sont ces informations qui permettent de guider le processus de recherche locale de telle manière que ces clauses soient satisfaites en priorité.

2.2.4 Stratégies d'échappement dédiées aux architectures GSAT et WSAT

Les différentes métaheuristiques présentées précédemment peuvent aisément être greffées aux architectures GSAT et WSAT. De plus, d'autres critères d'échappement ont été développées afin d'accroître l'efficacité de ces deux approches. Ces stratégies sont pour la plupart issue de la littérature concernant la recherche locale pour SAT. Nous présentons une liste non exhaustive de ces différents critères d'échappement parmi les plus connus :

- **WALKSAT** : cette stratégie, introduite par Selman *et al.* (1994), est caractérisée par la manière dont la prochaine variable à flipper est choisie. Les variables sont tout d'abord classées suivant le nombre de clauses falsifiées qu'elles génèrent si elles sont flippées. S'il existe une variable telle que inverser sa valeur de vérité ne falsifie aucune nouvelle clause alors cette variable est flippée. Sinon, avec une probabilité p, une variable est flippée de manière aléatoire et, avec une probabilité $1-p$, une variable est choisie de manière aléatoire parmi les variables dont le flip falsifie le moins de nouvelles clauses ;
- **NOVELTY** : cette stratégie est une variante de RANDOMWALK, elle classe (comme cette dernière) les variables suivant le nombre de clauses falsifiées auxquelles elles appartiennent. Soient la meilleure et la seconde meilleure variable selon cet ordre. Si la meilleure variable n'est pas la plus récemment flippée, alors elle est choisie. Sinon, avec une probabilité p, on choisit la seconde meilleure variable, et avec une probabilité $1-p$, on sélectionne la meilleure. Comparativement à la stratégie *Random Walk*, dans cette approche, la notion de choix aléatoire d'une variable selon une certaine probabilité a été supprimée au profit d'un choix non-optimal, mais satisfaisant puisqu'il s'agit de la variable qui lui succède dans l'ordre. De nombreuses variantes de NOVELTY ont été proposées (telles que RNOVELTY ou NOVELTY+), l'ensemble de ces stratégies est présenté et étudié dans (McAllester *et al.* 1997) ;
- **G^{2}WSAT** : la procédure G^{2}WSAT (*Gradient based Greedy WalkSat*) proposé par Li et Huang (2005) est basée sur la notion de « *promising decreasing variable* » et de « *promising decreasing path* ». Formellement, une variable est dite *decreasing* si elle permet d'améliorer la valeur de la fonction objectif lorsqu'elle est flippée. Soit x et y deux variables telles que $x \neq y$ et y n'est pas une variable *decreasing*. La variable y est dite *promising decreasing variable* si, après avoir flippé la variable x, la variable y devient *decreasing*. La notion de *promising decreasing path* est alors définie comme une séquence de flips telle que toutes les variables sont des *promising decreasing variables*.
  La méthode proposée consiste alors à choisir une variable dans l'ensemble des variables appartenant à l'ensemble des *promising decreasing variables* tant que cet ensemble n'est pas vide (définissant ainsi un *promising decreasing path*). Sinon, la procédure utilise un critère d'échappement classique pour choisir la prochaine variable à flipper (NOVELTY++ dans le papier présentant l'approche (Li et Huang 2005)) ;
- **SAPS** : cette méthode, introduite par Hutter *et al.* (2002), est basée sur le principe initié par Morris (1993) et qui consiste à s'échapper des minima locaux à travers la redéfinition de la fonction objectif. En plus d'incrémenter les poids des clauses falsifiées par chaque interprétation parcourue, des opérations supplémentaires sont effectuées sur les poids des clauses. En effet, lorsqu'un minimum local est rencontré, avec une probabilité p le poids de chaque clause est divisé par une constante, de manière à accentuer l'importance des dernières configurations rencontrées.

Comme nous l'avons souligné précédemment, ces stratégies d'échappement sont pour la plupart issues du domaine SAT et, à notre connaissance, elles n'ont pas toutes été étudiées dans le cadre de CSP. Nous proposons dans le chapitre 8 un solveur de recherche locale basé sur les architectures WSAT et GSAT afin d'implémenter et d'étudier le comportement de certaines d'entre elles.

Comme nous pouvons le remarquer, les méthodes utilisées afin de sortir des minima locaux sont tributaires d'un certain nombre de paramètres (redémarrage, taille de la liste tabou, probabilité de flipper aléatoirement, *etc.*). Pour des instances inconnues, une valeur optimale de ces paramètres ne peut être connue *a priori*. Afin de pallier ce problème, Hoos (2002) a proposé une approche permettant de les régler de manière dynamique en cours de recherche (cas de l'utilisation d'une approche utilisant une probabilité p comme pour l'utilisation du critère d'échappement NOVELTY ou RANDOMWALK). Cette méthode est présentée dans la section suivante.

2.3 Ajustement dynamique des paramètres d'un solveur de type recherche locale

Toutes les métaheuristiques s'appuient sur un equilibre entre l'intensification de la recherche et la diversification de celle-ci. D'un coté, l'intensification permet de rechercher des solutions de plus grandes qualités en s'appuyant sur les solutions déjà trouvées et de l'autre, la diversification met en place des stratégies qui permettent d'explorer un plus grand espace de solutions et d'échapper à des minima locaux. Ne pas préserver cet equilibre conduit à une convergence trop rapide vers des minima locaux (manque de diversification) ou à une exploration trop longue (manque d'intensification).

Comme indiqué précédemment, les métaheuristiques utilisées dans le cadre de la recherche locale dépendent souvent d'au moins un paramètre. Ce paramètre, excepté dans le cas de l'utilisation de la méthode tabou (taille de la liste tabou), représente une probabilité qui sera notée p par la suite et qui est une valeur comprise entre 0 et 1. Cette valeur a un impact important sur les performances du solveur. En particulier, il a été observé qu'une valeur suffisamment élevée de p diminuait ou annulait l'impact des redémarrages dans les solveurs de type WSAT (Parkes et Walser 1996, Hoos 1999) et qu'à l'inverse une valeur trop basse impose un réglage minutieux du paramètre contrôlant le redémarrage afin de pouvoir obtenir de bonnes performances (Hoos et Stützle 2000). Trouver la valeur optimale pour ce paramètre n'est pas une tâche aisée. Il est montré, de manière expérimentale (Hoos 2002), qu'une faible déviation (vis-à-vis de l'optimal) avait une incidence dramatique sur les résultats obtenus. En effet, la valeur optimale dépend énormément du type d'instances considérées, ce qui suggère une étude expérimentale fine afin de définir le meilleur réglage.

L'approche proposée par Hoos (2002) est basée sur l'idée qu'il apparaît raisonnable de supposer qu'un réglage optimal du paramètre contrôlant la diversification établit un bon compromis entre l'agilité du solveur à atteindre une bonne solution et l'agilité pour s'extirper des minima locaux. Le schéma proposé par l'auteur commence par effectuer uniquement une recherche gloutonne ($p = 0$). Cette phase atteint alors (hormis si une solution est trouvée) une stagnation de la recherche dans un minimum local. Lorsque cette situation se produit il est nécessaire d'augmenter la valeur de p afin de diversifier la recherche et ainsi pouvoir s'arracher du minimum locale.

Après avoir augmenté la valeur de p, il est nécessaire de détecter si le solveur s'est suffisamment éloigné du minimum local. Pour cela, l'évolution de la valeur de la fonction objectif est étudiée. Si, pendant un certain nombre d'étapes, la valeur de la fonction objectif n'est pas améliorée la variable p est une nouvelle fois augmentée (puisqu'il n'a pas encore été possible de s'extirper suffisamment du minimum local).

Au bout d'un certain temps, le bruit généré par l'augmentation de la variable p est suffisamment important pour permettre au solveur de sortir du minimum local. Une fois survenu, la valeur de p est graduellement diminuée, replaçant ainsi le solveur dans une phase d'intensification. Formellement, étant donnés deux paramètres Φ et Θ, le paramètre p est dynamiquement réglé de la manière suivante :

- si la recherche n'a pas permis d'améliorer la valeur de la fonction objectif pendant un certain nombre d'étapes égal à $\Theta \times |\mathcal{F}|$, alors $p = p + (1 - p) \times \Phi$;
- sinon, à chaque fois que la valeur de la fonction objectif est améliorée, p est ajusté tel que $p = p - p \times \Phi/2$.

Il apparaît, à première vue, que cette méthode transfère le problème lié au fait de régler le paramètre p à un problème peut-être plus difficile qui consiste à régler les paramètres Φ et Θ. Au final, l'ajout de ces deux paramètres ne pose pas réellement de problème. En effet, dans l'article où cette méthode est décrite, les auteurs fixent Φ à 0.2 et Θ à $1/6$ pour l'ensemble de leurs expérimentations Hoos (2002) et montrent que l'ajout de leur schéma permet d'obtenir une amélioration notable. En fait l'avantage est que bien que ces valeurs soient fixées, l'utilisation de ce schéma permet d'ajuster de manière dynamique le processus de diversification pendant la recherche.

Bien que initialement étudié dans le cadre du problème SAT ce schéma a l'avantage de pouvoir aisément être adapté au réglage d'autres paramètres et de pouvoir être utilisé dans d'autres contextes. En effet, les approches de type recherche locale ont toutes le même comportement et sont toutes mues par le même principe d'intensification et de diversification. Il semble donc assez logique d'étendre ce principe à l'ensemble des schémas dépendant de paramètres afin d'améliorer leurs efficacités (SAPS $\rightarrow$ RSAPS (Hutter *et al.* 2006), G^{2}WSAT $\rightarrow$ ADAPTG^{2}WSAT (Li *et al.* 2007), *etc.*). Nous verrons, dans la suite de ce manuscrit, deux de nos contributions concernant l'hybridation de méthodes (hybridation entre la recherche locale et une approche systématique) où ce principe est appliqué afin de gérer de manière efficace le passage d'une approche à une autre.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les algorithmes et les éléments nécessaires à l'élaboration de méthodes de recherche locale efficace pour la résolution pratique des problèmes de satisfaction de contraintes SAT et CSP. Ces algorithmes, quoique conceptuellement simples, permettent de résoudre un grand nombre de problèmes différents et sont sur certaines classes de problèmes les meilleurs stratégies de résolution. L'inconvénient est que ces modes de résolution sont incomplets et que par conséquent ils restent incapables de traiter des problèmes ne possédant pas de solution. Dans ce contexte, seule une approche assurant le parcours exhaustif de l'espace de recherche permet de conclure. Ces méthodes dites complètes font l'objet du chapitre suivant.

Chapitre 3

Approches complètes

Sommaire

CONTRAIREMENT à la recherche locale présentée précédemment, les algorithmes complets permettent, en un temps fini, de déterminer la consistance de n'importe quelle formule exprimée sous forme CNF ou sous un réseau de contraintes. Ils s'appuient généralement sur le parcours en profondeur d'un arbre de recherche, où chaque nœud correspond à l'assignation d'une variable et chaque chemin correspondant à une interprétation partielle des variables de la formule. L'objectif est alors de déterminer un chemin de la racine à une feuille - lequel représente une interprétation complète des variables de la formule - qui satisfait l'ensemble des contraintes du problème. Afin de ne pas explorer l'intégralité de cet arbre, la plupart des approches utilisent des méthodes de propagations de contraintes dans le but d'élaguer l'arbre de recherche en coupant les branches ne pouvant conduire à une solution.

Ce chapitre est décomposé en deux sections, lesquelles traitent respectivement de la résolution du problème SAT et du problème CSP à l'aide d'algorithmes complets.

3.1 Approche complète pour la résolution pratique du problème SAT

De nombreux algorithmes complets ont été proposés dans la littérature. Ces algorithmes se divisent principalement en deux catégories regroupant chacune plusieurs familles d'algorithmes. La première catégorie regroupe les familles d'algorithmes syntaxiques. Ces derniers s'intéressent à la structure du problème en ne prenant pas en compte la valeur des variables (résolution (Robinson 1965, Galil 1977), réécriture (Boole 1854, Shannon 1940), etc.). La seconde catégorie regroupe les familles d'algorithmes sémantiques qui, contrairement aux approches syntaxiques, s'attachent au sens des variables pour tenter de trouver une solution (énumération (Quine 1950, Davis *et al.* 1962, Jeroslow et Wang 1990), diagrammes de décision binaires (Akers 1978, Bryant 1992, Uribe et Stickel 1994), etc.).

En pratique, les approches les plus utilisées sont : les algorithmes énumératifs reposant le plus souvent sur la procédure de Martin Davis, Hilary Putnam, George Logemann et Donald Loveland (Davis *et al.* 1962), abrégée en DPLL et les algorithmes basés sur le principe de résolution. De plus, il est important de noter que, ces deux paradigmes sont à la base des solveurs SAT modernes. Ces solveurs, nommés CDCL (*Conflict Driving Clause Learning*) (Moskewicz *et al.* 2001a), sont en fait une combinaison fine de la procédure DPLL et d'une méthode basée sur le principe de résolution.

Après avoir introduit la résolution de Robinson (1965) et un ensemble d'approches s'appuyant sur ce principe, nous présentons une approche énumérative basée sur la notion d'arbre sémantique ainsi que les améliorations apportées à celle-ci et qui ont conduit à l'approche DPLL (algorithme de Quine (1950), simplification et heuristique de choix de variable). Nous terminons ce tour d'horizon des méthodes complètes pour SAT par la présentation des solveurs SAT modernes (CDCL) et des différents composants participant à leur efficacité (redémarrage, heuristique de choix de variable VSIDS et apprentissage).

3.1.1 Un algorithme syntaxique : le principe de résolution

Le principe de résolution de Robinson (1965) est l'application la plus connue du théorème de Herbrand [13]. Ce principe est élémentaire mais fondamental en logique propositionnelle. Appliqué à un ensemble de clauses, les deux règles suivantes plus la règle de subsumption (voir 1.2.3) permettent d'établir une méthode de démonstration automatique complète pour la réfutation.

Soit Σ un ensemble de clauses :
- **Règle de résolution** : si deux clauses α_1 et α_2 appartenant à Σ se résolvent en un littéral ℓ, et que la résolvante n'est pas tautologique, alors ajouter à Σ la résolvante γ en ℓ de α_1 et α_2 ;
- **Règle de fusion** : si $\alpha = \{x_1, x_2, \ldots, x_n, \ell, y_1, y_2, \ldots, y_m, \ell, z_1, z_2, \ldots, z_p\}$ est une clause de Σ, alors remplacer α par $\alpha_f = \{x_1, x_2, \ldots, x_n, \ell, y_1, y_2, \ldots, y_m, z_1, z_2, \ldots, z_p\}$;

Cette méthode consiste, à appliquer l'une de ces règles un certain nombre de fois, en enrichissant chaque fois l'ensemble de départ avec le résultat obtenu, le processus se termine soit par l'obtention d'une clause vide, auquel cas Σ est insatisfiable, soit par le fait qu'aucune clause nouvelle ne puisse plus être produite, auquel cas Σ est satisfiable. Lorsque plus aucune clause ne peut être produite, on dit que l'ensemble de clauses est saturé ou que l'on a effectué la saturation de celui-ci. Cette méthode est en fait basée sur le théorème suivant :

Théorème 3.1. *Soient deux clauses α_1 et α_2, si α_1 et α_2 se résolvent, alors la résolvante de α_1 et α_2 est une conséquence logique de α_1 et α_2.*

Par ailleurs, toute dérivation par résolution de la clause vide ($\alpha = \bot$) à partir d'une formule Σ est appelée *réfutation* ou *preuve*.

Définition 3.1 (dérivation par résolution et réfutation)**.** *Soient Σ une formule* CNF *et α une clause, on dit que α est dérivée de Σ par résolution s'il existe une séquence $[\beta_1, \beta_2, \ldots, \beta_n = \alpha]$ telle que $\forall 1 < i \leq n$ soit (i) $\beta_i \in \Sigma$, soit (ii) $\beta_i = \eta[x, \beta_k, \beta_j]$ avec $1 < j, k < i$.*

Il est commun de représenter la séquence de dérivation par un arbre inversé. L'arborescence s'appelle *arbre de dérivation*, ou *arbre de réfutation* dans le cas particulier où la clause vide est dérivée.

Définition 3.2 (arbre de résolution)**.** *Soient Σ un ensemble de clauses et α la clause que l'on souhaite dériver. Un arbre de résolution est un arbre tel que :*

13. Herbrand (1968) montre que pour qu'une formule Σ soit satisfiable, il faut qu'il existe un modèle dans Σ à vrai.

– *chaque feuille est étiquetée par une clause de* Σ *;*
– *chaque nœud qui n'est pas une feuille a deux fils et est étiqueté par une résolvante des clauses qui étiquette ses fils ;*
– *la racine de l'arbre est la clause* α*.*

Exemple 3.1. *Soient* $\Sigma = \{(p \vee \neg r \vee \neg t), (t \vee \neg q), (q \vee r)\}$ *et* $\alpha = (p \vee t \vee \neg q)$*. La Figure 3.1 représentée sous forme arborescente la séquence de dérivation* $[(p \vee \neg r \vee \neg t), (t \vee \neg q), (p \vee \neg r \vee \neg q), (q \vee r), (t \vee r), (p \vee t \vee \neg q)]$.

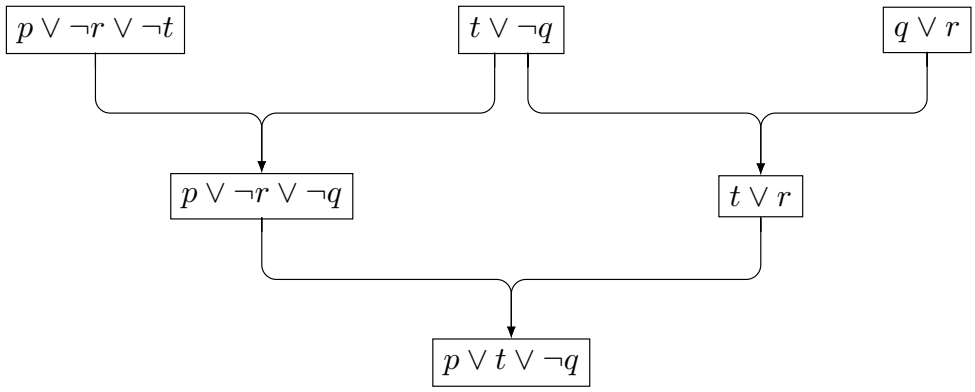


FIGURE 3.1 – Arbre de résolution.

Ce principe de résolution est en pratique relativement inefficace car le nombre de résolvantes à effectuer est souvent exponentiel, l'espace mémoire requis devient donc très important et les temps de calculs prohibitifs. Néanmoins, Chatalic et Simon (2000) montrent qu'un algorithme basé sur la résolution reste intéressant. Il obtiennent en effet des résultats comparables aux meilleurs prouveurs de l'époque sur certaines classes d'instances structurées.

De nombreuses stratégies ont été proposées afin de limiter l'explosion combinatoire du nombre de résolvantes, parmi elles citons :

– **la résolution de Davis et Putnam** (1960) : la méthode est basée sur l'application de la résolution comme processus d'élimination de variables. Concrètement, étant donnée une formule Σ fournie en entrée, l'algorithme commence par vérifier si la formule n'est pas trivialement satisfaite ($\Sigma = \emptyset$) ou falsifiée ($\bot \in \Sigma$). Si aucun de ces cas ne se présente, l'algorithme sélectionne une variable x de Σ, génère toutes les résolvantes en utilisant x puis supprime toutes les clauses contenant une occurrence de cette variable (la formule ainsi obtenue est équivalente du point de vue de la satisfiabilité). L'algorithme itère jusqu'à produire la formule vide (Σ est SAT), ou une formule contenant la clause vide (Σ est UNSAT). Un de ces cas se produit nécessairement puisqu'à chaque étape le sous-problème généré contient une variable en moins ;
– **ensemble support** (Wos *et al.* 1965) : le point de départ consiste à diviser la formule initiale Σ en deux ensembles de clauses Ψ et $\Sigma \setminus \Psi$ de telle sorte que $\Sigma \setminus \Psi$ soit satisfiable. L'ensemble de clauses Ψ^i_Σ pouvant être obtenu à l'étape i ($i > 0$) est tel que $\alpha \in \Psi^i_\Sigma$ si et seulement si $\alpha \in \Psi^{i-1}_\Sigma$ ou s'il existe $\beta \in \Psi^{i-1}_\Sigma$ et $\gamma \in \Sigma \cup \Psi^{i-1}_\Sigma$ tels que α est obtenue par résolution entre β et γ et $\Psi^0_\Sigma = \Psi$. L'ensemble Ψ est appelé ensemble support. Les auteurs montrent que si la formule Σ est insatisfiable alors il existe un i tel que Ψ^i_Σ contient la clause vide. Cette restriction est efficace si l'ensemble $\Sigma \setminus \Psi$ est le plus grand possible ;
– **la résolution régulière** (Davis *et al.* 1962, Tseitin 1968, Galil 1977) : le principe de cette approche repose sur la construction d'un arbre de preuve par résolution régulière. Un arbre de résolution est

dit régulier s'il n'existe pas de chemin d'une feuille à la racine où une variable est éliminée plus d'une fois par résolution. L'instance est prouvée insatisfiable si et seulement s'il existe un arbre de preuve par résolution régulier ;

- **la P-résolution et la N-résolution** (Robinson 1983) : si une clause participant à la résolution contient uniquement des littéraux positifs (respectivement négatifs) alors l'étape de résolution est appelée P-résolution (respectivement N-résolution). Si la P-résolution est utilisée (respectivement N-résolution), seules les résolvantes faisant intervenir des clauses positives (respectivement négatives) sont ajoutées. Nous obtenons un gain dans le processus de résolution dans le sens où le nombre de clauses strictement positives (respectivement négatives) est minime. Cependant, il est possible de trouver des formules qui se résolvent en temps polynomial en utilisant la résolution en général et qui utilisent un temps exponentiel en tenant compte de cette restriction. Malgré cela cette technique reste complète ;
- **la résolution linéaire** : cette méthode a été indépendemment proposée par Loveland (1970), Luckham (1970) et Zamov et Sharonov (1969). Cette approche est un raffinement de la résolution classique permettant de réduire significativement le nombre de résolutions redondantes produites. Le principe consiste à restreindre la séquence de résolution afin de ne considérer que des dérivations linéaires. Une dérivation linéaire Δ, d'un ensemble de clauses Σ, est une séquence de clauses $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ telles que $\alpha_1 \in \Sigma$ et chaque α_{i+1} est une résolvante de α_i (le plus proche parent de α_{i+1}) et d'une clause β telle que (i) $\beta \in \Sigma$, ou (ii) β est un ancêtre α_j de α_i où $j < i$;
- **la résolution unitaire** (Dowling et Gallier 1984) : la résolution unitaire, appelée aussi préférence unitaire, est un cas particulier de la résolution linéaire où une des clauses à résoudre est une clause unitaire. Bien évidemment cette méthode n'est pas complète ;
- **la résolution par entrée** (Chang et Lee 1973) : la résolution par entrée linéaire est un cas particulier de la résolution linéaire, où une clause à résoudre appartient toujours à l'ensemble initial de clauses. C'est une stratégie très efficace, mais elle n'est pas complète. Cette méthode se trouve à la base du fonctionnement du mécanisme de démonstration des théorèmes du langage Prolog ;
- **la résolution étendue** (Tseitin 1983) : c'est une extension de la résolution à laquelle on ajoute la règle suivante : à chaque étape de la construction de la preuve, il est possible d'ajouter des lemmes à la formule, sous la forme d'une nouvelle variable y associée aux clauses qui codent $y \Leftrightarrow (\ell_1 \vee \ell_2)$ (*ie.* $(y \vee \ell_1), (y \vee \ell_2)$ et $(\neg y \vee \neg \ell_1 \vee \neg \ell_2)$), où ℓ_1 et ℓ_2 sont deux littéraux apparaissant précédemment dans la preuve. Tout en étant d'apparence une règle très simple, l'ajout de nouvelles variables permet à la résolution étendue d'être très efficace (Cook 1976) ;
- **la résolution sémantique** (Dennis 1994) : la résolution sémantique est une extension de la résolution classique utilisant un modèle ou une interprétation afin de guider la recherche de preuve par réfutation. Plusieurs variantes de la résolution sémantique existent (pour un aperçu complet voir l'article de Dennis (1994)) ;
- **la résolution restreinte** : cette approche consiste à ajouter à la formule un certain nombre de clauses produites par la résolution et à lancer un solveur basée sur la résolution régulière ensuite (ce dernier étant dans la plupart des cas beaucoup plus efficace que le principe de résolution). Diverses restrictions peuvent être mises en place. Ces restrictions sont plus ou moins complexes et plus ou moins bénéfiques pour DPLL (voir par exemple (Génisson et Siegel 1994, Castell 1996, Billionnet et Sutter 1992)) ;
- **la résolution dirigée** (Dechter et Rish 1994) : la méthode commence par partitionner la formule initiale Σ en plusieurs sous ensembles $\Omega_1, \Omega_2, \ldots, \Omega_n$ suivant un ordre $\mathcal{O} = \{x_1, x_2, \ldots, x_n\}$ pris sur l'ensemble des variables $\mathcal{V}_\Sigma$ de la formule initiale. Chaque Ω_i contient l'ensemble des clauses dont le plus grand (pour notre ordre $\mathcal{O}$) littéral est x_i. Une fois cette partition établie, l'ensemble des partitions Ω_i sont considérées dans l'ordre. Alors, pour chaque paire de clauses $\{(\alpha \vee x_i), (\beta \vee \neg x_i)\} \subseteq \Omega_i$: si $\gamma = \alpha \vee \beta$ est une clause vide alors le problème est insatisfiable

sinon la clause γ est ajoutée dans une partition Ω_j telle qu'il n'existe pas de littéral y dans γ avec y plus grand que x_j pour l'ordre $\mathcal{O}$ considéré.

Dans la pratique, ces méthodes sont très rarement utilisées dans leur forme originale. Cependant, nous verrons dans la suite de ce manuscrit (voir 3.1.5.1) que le principe de résolution est une composante très importante des solveurs SAT modernes. Qui plus est, nous pouvons aussi noter qu'une forme limitée de DP est à la base d'un des meilleurs pré-traitements de formules CNF (voir 3.1.5.4) et est intégrée dans la plupart des solveurs SAT modernes. Cette forme limitée applique la résolution pour éliminer une variable uniquement si la taille de la formule n'augmente pas. En pratique, cette technique permet d'éliminer un nombre non négligeable de variables lorsqu'elle est appliquée sur des instances codant des applications réelles.

3.1.2 Arbres sémantiques

Contrairement aux approches syntaxiques telles que la résolution, les méthodes énumératives se situent au niveau sémantique, c'est-à-dire au niveau de l'interprétation des formules. Pour montrer qu'un ensemble de clauses est incohérent (respectivement cohérent), les méthodes énumératives énumèrent les interprétations de la formule et montrent qu'elles sont des contre-modèles (respectivement qu'au moins l'une d'elles est un modèle) de cet ensemble de clauses. L'idée de base est la construction d'un arbre binaire de recherche où chaque nœud représente une interprétation partielle des variables propositionnelles, apparaissant dans l'ensemble des clauses. La méthode des arbres sémantiques repose sur la construction d'un arbre binaire de recherche.

Définition 3.3 (arbre sémantique)**.** *Soit Σ une formule* CNF*, contenant l'ensemble des variables propositionnelles $\mathcal{V}_\Sigma = \{x_1, x_2, \ldots, x_n\}$, l'arbre sémantique correspondant à Σ est construit comme suit :*

- *chaque arc est étiqueté par un littéral x_i ou $\neg x_i \in \mathcal{V}_\Sigma$. La branche x_i correspond à l'affectation de x_i à vrai et la branche $\neg x_i$ correspond à l'affectation de x_i à faux ;*
- *les littéraux étiquetant les arcs issus d'un même nœud sont opposés ;*
- *aucune branche ne comporte plus d'une occurrence de chaque variable.*

Chaque chemin de la racine à un nœud de l'arbre correspond à une interprétation partielle de Σ.

Définition 3.4 (arbre complet)**.** *Soit $\mathcal{V}_\Sigma$ l'ensemble des variables propositionnelles apparaissant dans un ensemble de clauses Σ. L'arbre sémantique correspondant à Σ est dit complet si et seulement si, en plus des conditions exigées sur les arbres sémantiques, chaque chemin de la racine à une feuille de l'arbre correspond à une interprétation complète de l'ensemble des variables de $\mathcal{V}_\Sigma$.*

Définition 3.5 (branche fermée)**.** *Une branche d'un arbre sémantique correspondant à une formule Σ est dite fermée si et seulement s'il existe un neoud N tel que l'interprétation partielle correspondante à celui-ci falsifie une des clauses de Σ et tel que les interprétations partielles de tout nœud ancêtre de N ne falsifient aucune clause de Σ.*

Définition 3.6 (arbre fermé)**.** *Un arbre sémantique correspondant à une formule Σ est dit fermé si et seulement si toutes les branches de cet arbre sont fermées.*

Pour prouver qu'une formule Σ est cohérente il suffit de trouver une feuille qui produit un modèle, c'est-à-dire une interprétation qui satisfait l'ensemble des clauses. Pour prouver l'incohérence d'un ensemble de clauses il suffit montrer que l'arbre est fermé. Comme l'arbre sémantique comporte 2^n feuilles cette méthode est assez inefficace en pratique.

Exemple 3.2. *Soit* $\Sigma = \{(a \vee b \vee \neg c), (\neg a \vee \neg b \vee c), (\neg c \vee d), (\neg c \vee \neg d), (a \vee \neg b \vee c), (\neg a \vee b \vee \neg c \vee d), (b \vee c), (\neg a \vee b)\}$ *un ensemble de clauses, l'ensemble des propositions est* $\mathcal{V}_\Sigma = \{a, b, c, d\}$ *et l'arbre sémantique correspondant à* Σ*, où l'heuristique de branchement utilisée est l'ordre lexicographique est illustré dans la figure 3.2.*

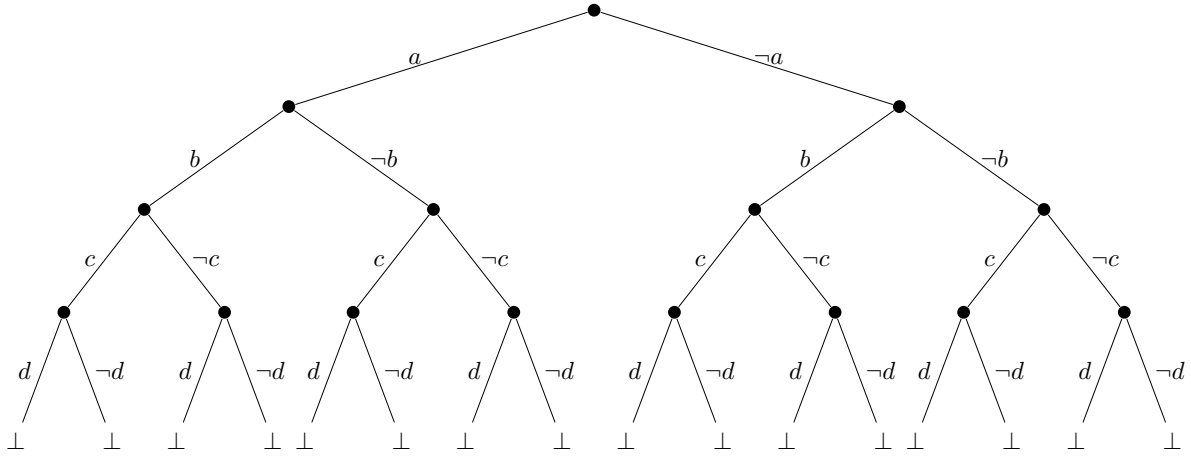


FIGURE 3.2 – Arbre sémantique correspondant à l'ensemble de clauses de l'exemple 3.2.

3.1.3 Méthode de Quine

L'algorithme de Quine (1950) est une amélioration de la méthode des arbres sémantiques dans laquelle on réalise à chaque nœud de l'arbre binaire une valuation partielle de la formule. Cette évaluation permet de simplifier la formule en tenant compte des assignations de variables déjà réalisées. Si une évaluation partielle permet de conclure directement à l'incohérence de la sous formule, alors il n'est pas nécessaire de poursuivre la construction de l'arbre au delà de ce nœud. Ainsi, le nombre de feuilles d'un arbre sémantique construit selon la méthode de Quine peut, dans certains cas, être nettement inférieur à ce que l'on obtiendrait en construisant systématiquement des arbres sémantiques complets. La méthode de Quine, décrit dans l'Algorihme 3.1, est basée sur la proposition suivante :

Proposition 3.2. Σ *est incohérent si et seulement si* $\Sigma_{|\ell}$ *et* $\Sigma_{|\neg\ell}$ *sont incohérent.*

Algorithme 3.1 : QUINE

Données : Σ un ensemble de clauses
Résultat : `vrai` si la formule est consistante, `faux` sinon

```
1 Début
2   si (Σ = ∅) alors retourner vrai;
3   si (⊥ ∈ Σ) alors retourner faux;
4   ℓ ⟵ HeuristiqueDeBranchement(Σ);
5   retourner (QUINE(Σ_|ℓ) ou QUINE(Σ_|¬ℓ))
6 Fin
```

Exemple 3.3. *Considérons la formule de l'exemple 3.2. L'arbre binaire de recherche construit implicitement par la méthode de Quine est illustré dans la figure 3.3.*

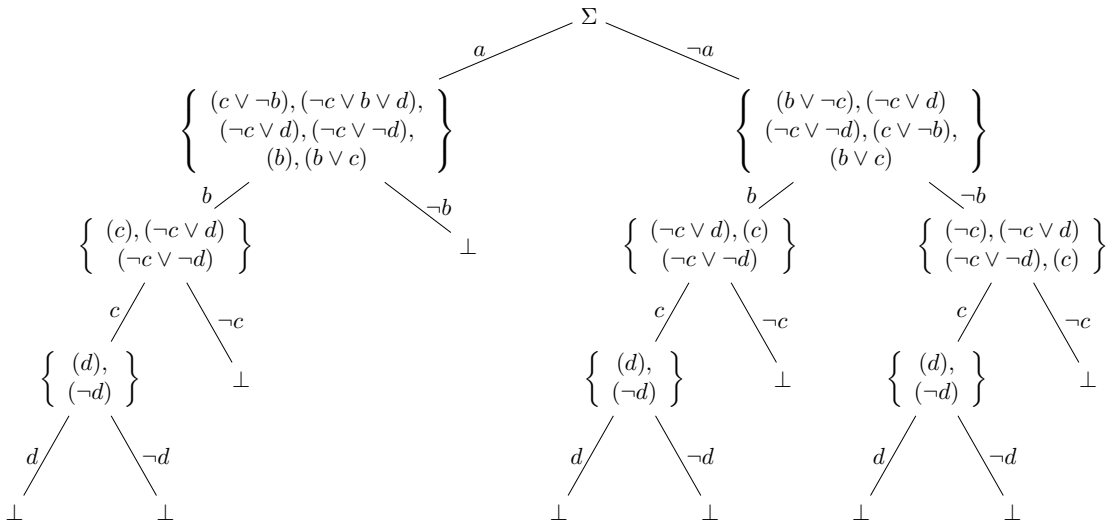


FIGURE 3.3 – Arbre construit par la méthode de Quine sur l'ensemble de clauses de l'exemple 3.2.

3.1.4 La procédure de Davis - Putnam - Logemann - Loveland

En 1960, Davis et Putnam (1960) proposent une amélioration de l'algorithme de Quine. Cet algorithme tire parti de la forme normale conjonctive des formules propositionnelles pour sélectionner au mieux les propositions, ceci afin d'éviter la construction de branches inutiles. Le principal inconvénient de cette approche est lié à la complexité spatiale de celle-ci. Pour palier ce problème, en 1962, Martin Davis, George Logemann et Donald Loveland (Davis *et al.* 1962) proposent une procédure appelée DPLL. Cette méthode est une amélioration de la procédure DP copiant une partie du modèle algorithmique proposé dans celui-ci, DPLL emprunte à DP :

- ses critères d'arrêt :
  - l'absence de clause indiquant que la formule est satisfiable ;
  - l'apparition de la clause vide signifiant que la formule n'admet pas de modèle ;
- ses simplifications par littéraux unitaires et purs [14] ;
- le fait de supprimer une variable de la formule à chaque étape, permettant ainsi de réduire la complexité du problème à traiter.

Ces emprunts expliquent certainement pourquoi les algorithmes DP et DPLL ont souvent été confondus dans la littérature et également pourquoi le nom de Hilary Putnam est également associé à DPLL alors qu'il ne figure pas dans la liste des auteurs de l'article introduisant DPLL (Davis *et al.* 1962). La méthode DPLL, décrit par l'algorithme 3.2, est en fait un algorithme de recherche arborescent de type « *depth-first search* » (recherche en profondeur d'abord) avec retour arrière. Étant donnée une formule Σ, cette procédure consiste à choisir un littéral ℓ de Σ (ligne 4) et à décomposer la formule Σ en deux sous-formules $\Sigma \wedge \ell$ et $\Sigma \wedge \neg\ell$. Ce principe, appelé *séparation*, est basé sur le fait que Σ est consistante si et seulement si $\Sigma \wedge \ell$ ou $\Sigma \wedge \neg\ell$ sont consistant. Une fois cette décomposition matérialisée, la procédure DPLL consiste à tester la validité de la première sous formule, si elle est consistante alors le problème est consistant, sinon la seconde formule est testée (ligne 5). Afin d'éviter l'exploration inutiles de branches de l'arbre de recherche, certaines simplifications sont opérées. La principale est la propagation unitaire (cf 3.1.4.1). D'autres processus de filtrages sont possibles (*hyperbin resolution* (Bacchus 2002), simplification par littéraux purs, *etc.*) mais sont très rarement applicables de manière efficace en pratique.

14. Un littéral ℓ d'une formule Σ est dit pur si $\nexists \alpha \in \Sigma$ telle que $\tilde{\ell} \in \alpha$.

Algorithme 3.2 : DPLL

Données : Σ un ensemble de clauses
Résultat : `vrai` si la formule est consistante, `faux` sinon
1 **Début**
2 $\quad \Sigma \longleftarrow$ `SIMPLIFICATION(`Σ`);`
3 $\quad$ **si** $(\Sigma = \emptyset)$ *ou* $(\bot \in \Sigma)$ **alors retourner** $(\Sigma = \emptyset)$ *ou* $(\bot \notin \Sigma)$;
4 $\quad \ell \longleftarrow$ `HeuristiqueDeBranchement(`Σ`)`;
5 $\quad$ **retourner** $($`DPLL(`$\Sigma \wedge \ell$`)` *ou* `DPLL(`$\Sigma \wedge \neg\ell$`)`$)$
6 **Fin**

La procédure DPLL a des propriétés théoriques intéressantes. En effet, cette approche est complète et cohérente dans le sens où chaque solution candidate retournée est une solution du problème initial. Une telle solution, si elle existe, est toujours découverte à condition de laisser à la méthode un temps suffisant pour répondre. D'un point de vue pratique cette procédure, dans le pire des cas, est exponentielle en temps par rapport à la taille du problème et (contrairement à la procédure DP) polynomiale en espace. Bien que les processus de filtrages permettent de réduire considérablement le nombre de variables à affecter, le temps de réponse de la procédure DPLL peut rester exponentiel (Crawford et Auton 1993).

Un point crucial quant à l'efficacité de la procédure DPLL est le choix de la variable pour la règle de séparation. Ce choix est fait via une heuristique. Cette *heuristique de branchement* a fait l'objet de nombreuses études et sera présentée plus en détail par la suite (cf 3.1.4.3). La variable ainsi choisie est communément appelée variable (ou littéral) de décision.

L'exemple ci dessous illustre le fonctionnement de l'algorithme DPLL sur l'instance de l'exemple 3.2. Les branches issues de littéraux unitaires et purs ne sont pas explorées.

Exemple 3.4. *Considérons la formule de l'exemple 3.2. L'arbre décrit par l'algorithme* DPLL *pour l'ensemble de clauses Σ est représenté dans la figure 3.4. Cet arbre n'admettant pas de feuilles représentant l'ensemble vide de clauses, Σ est démontrée inconsistante.*

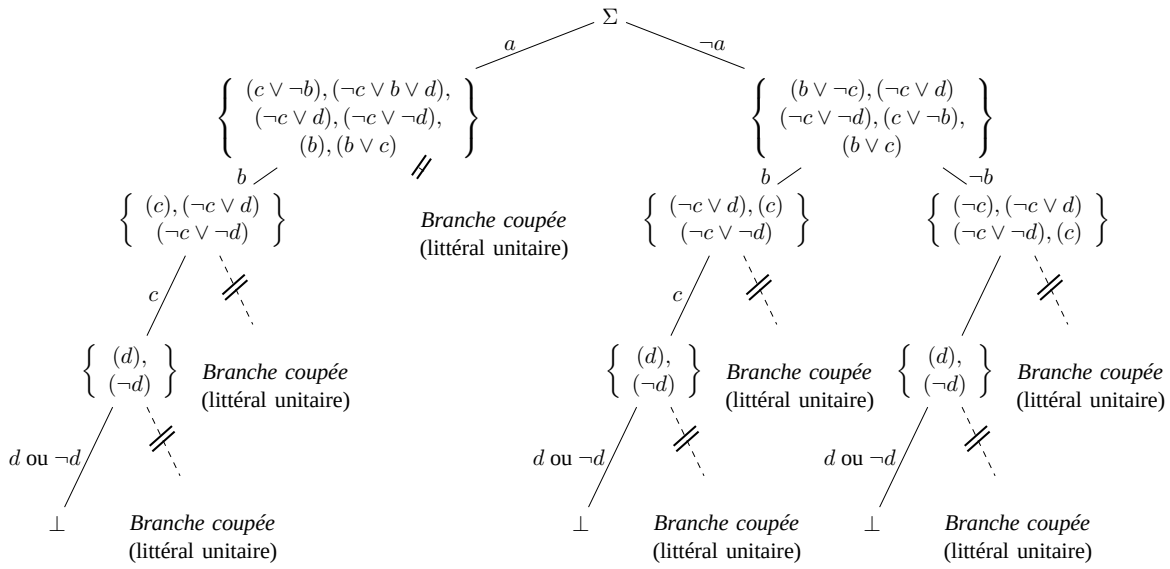


FIGURE 3.4 – Arbre de recherche construit par l'algorithme DPLL sur l'instance de l'exemple 3.2.

Nous pouvons clairement voir sur la Figure 3.4 l'intérêt des méthodes de simplifications. En effet, l'utilisation de la propagation unitaire et de la simplification par les littéraux purs permet d'éviter le parcours d'un nombre important d'interprétations inutiles.

3.1.4.1 La propagation unitaire

La propagation des contraintes booléennes ou résolution unitaire ou encore propagation unitaire représente la forme de simplification la plus utilisée et sans doute la plus utile des approches de type DPLL. En effet, sur de nombreuses instances la plupart des affectations (plus de 90%) sont effectuées par le biais de ce mécanisme. Cela explique les nombreuses améliorations théoriques et pratiques de la propagation unitaire (Moskewicz *et al.* 2001a, Zhang et Stickel 1996). La propagation des contraintes booléennes repose sur la propriété élémentaire suivante :

Propriété 3.3. *Soit Σ une formule* CNF. *Si ℓ est un littéral unitaire de Σ, alors Σ est satisfiable si et seulement si $\Sigma_{|\ell}$ est satisfiable.*

Cette propriété découle du fait que les seules interprétations potentiellement capable de satisfaire la formule doivent satisfaire les littéraux appartenant aux clauses unitaires. En effet, satisfaire le littéral opposé de l'un d'eux conduit nécessairement à une clause vide et donc à une contradiction. Simplifier par un littéral unitaire consiste donc à supprimer de l'ensemble de clauses, toutes celles contenant le littéral unitaire et à supprimer toutes les occurrences du littéral complémentaire, c'est-à-dire « raccourcir » les clauses contenant le littéral complémentaire. La propagation unitaire est l'application répétée de cette simplification jusqu'à ce que la base de clauses ne contienne plus de clauses unitaires (point fixe) ou jusqu'à l'obtention d'une clause vide (contradiction). Formellement, nous avons :

Définition 3.7 (propagation unitaire)**.** *Soit Σ une formule, nous notons Σ^* la fermeture par propagation unitaire de Σ. Σ^* est définie récursivement comme suit :*
- *$\Sigma^* = \Sigma$ si $\nexists \alpha \in \Sigma$ telle que α est unitaire ;*
- *$\Sigma^* = \bot$ si Σ^* contient les deux clauses unitaire (ℓ) et $(\neg\ell)$;*
- *$\Sigma^* = (\Sigma_{|\ell})^*$ tel que ℓ est le littéral apparaissant dans une clause unitaire de Σ.*

Propriété 3.4. *Un ensemble de clauses Σ est consistant si et seulement si Σ^* est consistant.*

Exemple 3.5. *Soit $\Sigma = \{(a \vee b \vee c \vee d), (a \vee \neg b \vee e), (\neg a \vee e \vee c), (\neg c \vee \neg e), (c)\}$, l'application de la propagation unitaire jusqu'à l'obtention d'un point fixe est dépeinte sur la Figure 3.5.*

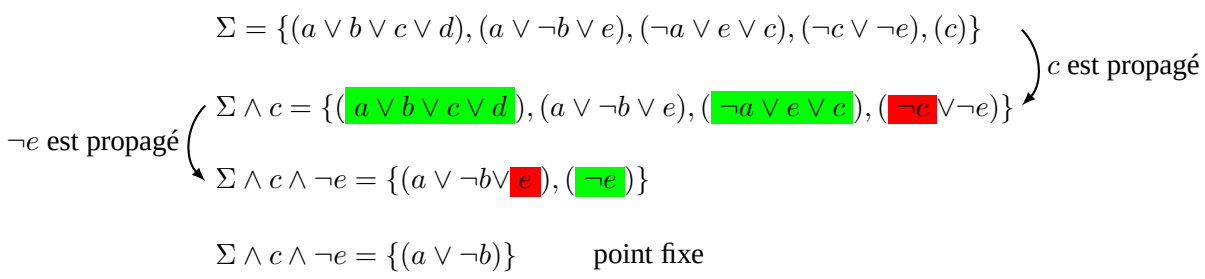


FIGURE 3.5 – Application de la propagation unitaire sur un ensemble de clauses.

L'application de la propagation unitaire au sein de l'algorithme DPLL implique une modification au niveau de la gestion des retours arrière (*backtracks*). En effet, lorsqu'un *backtrack* est effectué il est

nécessaire de considérer en plus du dernier point de choix les propagations unitaires qu'il a généré. Pour effectuer cela il est nécessaire d'utiliser une notion de pile de propagations où chaque élément de cette pile représente une séquence de décisions-propagations.

Définition 3.8 (séquence de propagations)**.** *Soit Σ une formule, $\mathcal{P} = \langle x_1, x_2, \ldots, x_n \rangle$ représente la séquence de propagations obtenue à partir de Σ telle que $\forall x_i \in \mathcal{P}$ la clause unitaire $(x_i) \in \Sigma_{|\{x_1, x_2, \ldots, x_{i-1}\}}$.*

Définition 3.9 (séquence de décisions-propagations)**.** *Soient Σ une formule et x un littéral de Σ, $\mathcal{S} = \langle (x), x_1, x_2, \ldots, x_n \rangle$ représente la séquence de décisions-propagations obtenue par l'application de la propagation unitaire sur la formule $(\Sigma \wedge x)$ telle que $\langle x_1, x_2, \ldots, x_n \rangle$ est une séquence de propagations obtenue à partir de $(\Sigma \wedge x)$.*

Définition 3.10 (pile de propagations)**.** *Soient Σ une formule et $\delta = [x_1, x_2, \ldots, x_n]$ une séquence d'affectations. La pile de propagations $\mathcal{H} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ obtenue par l'application de la propagation unitaire sur la formule Σ en fonction de la séquence δ est définie récursivement de la manière suivante :*

- *$\mathcal{S}_0$ est une séquence de propagations obtenue à partir de Σ ;*
- *$\mathcal{S}_1$ est une séquence de décisions-propagations obtenue à partir de Σ^* en considérant x_1 ;*
- *$\forall 1 < i \leq n, \mathcal{S}_i$ est une séquence de décisions-propagations obtenue à partir de $(\Sigma \wedge \bigwedge_{j=1}^{i-1} x_j)^*$ en considérant le littéral x_i.*

Exemple 3.6. *Soient $\Sigma = \{(\neg a \vee b), (\neg b \vee c), (\neg d \vee \neg e), (e \vee f), (e \vee \neg f \vee \neg g), (\neg h \vee \neg i), (a), (l \vee \neg e)\}$ une formule et $\delta = [d, h]$ une séquence de décisions. La pile de propagations obtenue après l'affectation des littéraux de la séquence de décisions est $\mathcal{H} = \{\langle a, b, c \rangle, \langle (d), \neg e, f, \neg g \rangle, \langle (h), \neg i \rangle\}$.*

Remarque 3.1. *Lorsqu'une séquence de décisions conduit à une contradiction le dernier niveau de la pile de propagation contient les deux littéraux complémentaires ayant amenés à celle-ci.*

Exemple 3.7. *Considérons la formule Σ de l'exemple 3.6 et la séquence de décisions conflictuelle $[g, \neg l]$. La pile de propagation obtenue est $\{\langle a, b, c \rangle, \langle (g) \rangle, \langle (\neg l), \neg e, f, \neg f \rangle\}$.*

Remarque 3.2. *Dans la suite de ce manuscrit nous ne faisons plus la distinction entre interprétation partielle issue de la propagation unitaire et la séquence de décisions-propagations associée.*

Exemple 3.8. *Considérons la formule et la séquence de décisions de l'exemple 3.6. L'interprétation partielle obtenue par propagation est $\mathcal{I} = \{a, b, c, (d), \neg e, f, \neg g, (h), \neg i\}$.*

À partir de la notion de pile de propagations nous pouvons introduire pour un littéral ℓ de cette pile les notions de niveau de propagation ($niv(\ell)$), de clause raison ($\overrightarrow{cla}(\ell)$) et d'explication ($exp(\ell)$).

Définition 3.11 (niveau de propagation)**.** *Soient Σ une formule, $\mathcal{H} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ une pile de propagations obtenue à partir de Σ en appliquant la séquence de décisions $[x_1, x_2, \ldots, x_n]$ et ℓ un littéral de Σ. Si $\exists \mathcal{S}_i \in \mathcal{H}$ tel que $\ell \in \mathcal{S}_i$ alors $niv(\ell) = i$, $niv(\ell) = \infty$ sinon.*

Exemple 3.9. *Considérons la formule et la séquence de décisions de l'exemple 3.6. Nous avons $niv(a) = 0$, $niv(\neg i) = 2$ et $niv(l) = \infty$.*

Définition 3.12 (clause raison)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation à partir de Σ en appliquant la séquence de décisions $\delta = [x_1, x_2, \ldots, x_n]$ et ℓ un littéral de Σ. Si $\ell \in \delta$ alors $\overrightarrow{cla}(\ell) = \bot$, sinon $\overrightarrow{cla}(l) \in \Sigma$ telle que $\ell \in \overrightarrow{cla}(\ell)$ et $\forall y \in \overrightarrow{cla}(\ell), \mathcal{I}(y) = \bot$ et y précède ℓ dans l'interprétation $\mathcal{I}$.*

Remarque 3.3. *Il est possible d'associer plusieurs clauses raison à un littéral propagé. Néanmoins, il est usuel de n'en considérer qu'une seule.*

Définition 3.13 (explication)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation à partir de Σ en appliquant la séquence de décisions $\delta = [x_1, x_2, \ldots, x_n]$ et ℓ un littéral de Σ. Si $\ell \in \delta$ alors $exp(\ell) = \emptyset$, sinon $exp(\ell) = \{\neg x$ tel que $x \in \overrightarrow{cla}(\ell)$ avec $\overrightarrow{cla}(\ell)$ une clause raison de $\ell\}$.*

Remarque 3.4. *Comme pour les clauses raison il n'y a pas unicité de l'explication de la propagation d'un littéral.*

Exemple 3.10. *Considérons la formule et la séquence de décisions de l'exemple 3.6. Nous avons $\overrightarrow{cla}(g) = (e \vee \neg f \vee \neg g)$ et $exp(g) = \{\neg e, f\}$.*

Pour terminer, il est important de souligner que, en plus d'être fondamentale pour l'algorithme DPLL et d'être implémentée dans tous les solveurs de ce type, la propagation unitaire peut également être utilisée comme méthode de prétraitement. L'application de la propagation unitaire comme prétraitement permet en pratique d'extraire des instances certaines informations importantes comme les littéraux impliqués, équivalents ou encore unitaires (Le Berre 2001, Novikov 2003). De plus, elle permet aussi de déduire des formules comme des portes logiques ou des sous-clauses (Génisson et Siegel 1994, Darras *et al.* 2005, Grégoire *et al.* 2005). La collecte de ces informations est souvent obtenue par l'application d'une forme affaiblie de la conséquence logique. Cette affaiblissement se définit formellement de la manière suivante.

Définition 3.14 (conséquence logique restreinte à la propagation unitaire)**.** *Soient Σ et Σ' deux formules. Σ' est une conséquence logique restreinte à la propagation unitaire de Σ, notée $\Sigma \models^* \Sigma'$ si et seulement si toute clause $\alpha \in \Sigma'$, $(\Sigma \wedge \overline{\alpha})^*$ contient la clause vide.*

Exemple 3.11. *Considérons la formule Σ de l'exemple 3.6, La clause $(l \vee \neg g)$ est impliquée par propagation unitaire à partie de Σ.*

Comme nous l'avons souligné, la propagation unitaire est un processus fondamental de l'algorithme DPLL. Généralement, le temps consacré à celui-ci est de l'ordre de 90% du temps CPU utile à la résolution d'une instance. C'est en partie pour cette raison que l'implémentation de celle-ci a connu quelques évolutions. Nous noterons la principale qui est les « *watched literals* ».

3.1.4.2 Les structures de données paresseuses « *watched two literals* »

Afin d'améliorer l'efficacité de la propagation unitaire, de nombreux progrès algorithmiques ont été effectués ces dernières années, mais le plus important reste incontestablement la méthode proposée par Zhang et Malik (2002). Les auteurs introduisent une structure de données dite « paresseuse » qui permet un traitement de la propagation unitaire d'une complexité moyenne sous-linéaire. L'idée sous-jacente est d'avoir, pour chaque clause α de la formule, deux littéraux x et y appartenant à α non affectés à faux par l'interprétation courante. Ces deux littéraux sont garant du fait que la clause n'est ni unitaire ni falsifiée. En effet, afin de savoir si une clause est unitaire ou non, il suffit de vérifier s'il y a plus d'un littéral non affecté à faux dans celle-ci. Lorsque les deux littéraux x et y de α ont été désignés pour surveiller la clause α il est nécessaire de le leur indiquer. Pour cela, chaque littéral possède une liste de clauses qu'il doit surveiller. De cette manière, lorsqu'un littéral x est propagé à vrai il suffit de regarder la liste des clauses surveillées par le littéral complémentaire $\tilde{x}$ et à chercher un autre « surveillant » pour chacune d'elles.

Exemple 3.12. *Soit* $\Sigma = \{\alpha_1 = (a \vee b), \alpha_2 = (a \vee \neg b \vee c), \alpha_3 = (a \vee \neg b \vee \neg c)\}$*, la figure 3.6 représente une attribution possible des clauses aux différents littéraux.*

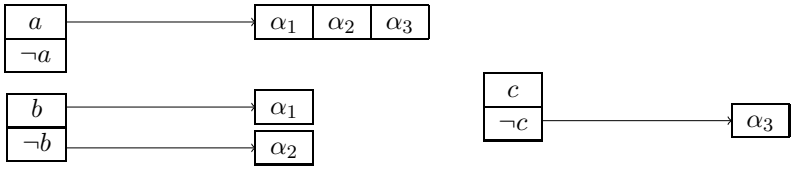
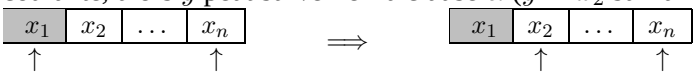


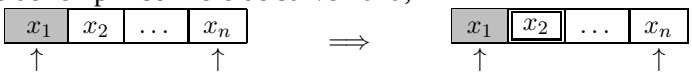
FIGURE 3.6 – Attribution de clauses aux différents littéraux.

À présent étudions les différents cas pouvant survenir lorsqu'un littéral est propagé à faux. Pour cela, considérons le cas d'une clause α surveillée par deux littéraux x_1 et x_n. En grisant, en entourant et en désignant respectivement les littéraux falsifiés, satisfait et marqués, les différents cas rencontrés lors de la recherche d'un nouveau littéral sentinelle, après l'affectation d'un littéral $\neg x_1$, sont les suivants :

– commençons par considérer le cas où il existe $y \in \alpha$ tel que $y \neq x_1$, $y \neq x_n$ et y est non affecté, par l'interprétation courante, alors y peut surveiller la clause α ($y = x_2$ sur la figure) ;

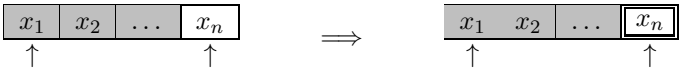


– supposons à présent que lors du processus de recherche d'un nouveau littéral non affecté la procédure « s'aperçoit » qu'il existe $y \in \alpha$ tel que y est satisfait par l'interprétation courante ($y = x_2$ sur la figure). Dans ce cas, il n'est pas utile de changer le rôle de surveillant pour x_1. En effet, puisque x_1 est affecté après y, lorsque y sera de nouveau « libre » x_1 le sera aussi et par conséquent il sera de nouveau à même de remplir son rôle de surveillant ;



– enfin, considérons le cas où il n'y a aucun littéral différent de x_n non affecté à faux. Dans cette configuration, deux situations se présentent :

1. x_n est non affecté et il est alors propagé à vrai ;



2. x_n est affecté à faux et alors la clause est falsifiée par l'interprétation courante. Dans ce cas, la propagation unitaire conduit à un conflit et un « *backtrack* » est nécessaire.

Pour les même raisons que celles énoncées dans le cas où un littéral est satisfait, il n'est pas nécessaire de changer les littéraux surveillant la clause.

Nous pouvons observer que, grâce à cette structure, lors de l'affectation d'une variable seul un sous-ensemble des clauses contenant une occurrence de cette variable est parcouru (voir figure 3.6). Pourtant ce traitement partiel est suffisant pour déclencher les propagations unitaires durant le parcours de l'espace de recherche. Ceci explique la rapidité de traitement des affectations, et en particulier de la propagation unitaire. De plus, il est important de noter qu'aucune mise à jour n'est nécessaire lors des retours arrière. Cependant, cette structure de données possède également quelques inconvénients : les clauses n'étant pas considérées dans leur intégralité, certains traitements tels que l'utilisation d'heuristiques syntaxiques ne sont plus possibles (cf. 3.1.4.3) car ils requièrent une connaissance complète de l'instance après affectation(s). Seules les heuristiques sémantiques peuvent être appliquées avec les structures de données paresseuses, la structure du problème n'étant pas connue.

Pour information, il est à noter que cette notion de « *watched literals* » peut facilement être étendue à d'autres situations où il est nécessaire de vérifier à chaque instant une condition. Par exemple, dans le cadre de la recherche locale, elle peut être utilisée afin de détecter efficacement lorsqu'une clause devient unisatisfaite ou falsifiée (Fukunaga 2004). De plus, l'utilisation de cette structure n'est pas non plus exclusivement réservée au problème SAT. En particulier, Gent *et al.* (2006) étendent ce principe au cadre CSP afin d'effectuer le maintien de l'arc cohérence généralisée (cf. 3.2.3.1).

3.1.4.3 Heuristiques de branchements

Comme nous l'avons souligné lors de la description de l'approche DPLL, le choix de la prochaine variable à affecter est un critère déterminant. Son incidence sur la taille de l'arbre et par conséquent les temps d'exécution sont considérables. En effet, l'arbre de recherche exploré peut varier de manière exponentielle en fonction de l'ordre avec lequel les variables sont affectées (Li et Anbulagan 1997). Cependant, sélectionner à chaque point de choix la variable qui permet de conduire à l'obtention d'un arbre de taille minimal est un problème *NP-difficile* (Liberatore 2000).

Au vu de la complexité théorique pour l'obtention de la variable de branchement optimale, il apparaît dès lors raisonnable d'estimer le plus précisément possible à l'aide d'une heuristique cette variable plutôt que de la calculer précisément. Cette heuristique, pour être efficace, doit permettre de diminuer la hauteur de l'arbre de recherche. De plus, un compromis entre le temps nécessaire au choix de la variable et le nombre de nœuds économisés doit être effectué. En effet, une heuristique trop gourmande en temps, même si elle réduit considérablement la taille de l'arbre, peut être moins performante qu'une heuristique beaucoup moins coûteuse en temps mais qui générera plus de nœuds.

Ces dernières années beaucoup d'heuristiques de branchements différentes ont été proposées. Nous distinguons en général trois types d'approches, les approches syntaxiques qui permettent d'estimer le nombre de propagations résultant de l'affectation d'une variable, les approches prospectives (de type « look-ahead ») qui permettent de détecter de futures situations d'échec, et les approches rétrospectives (de type « look-back ») qui essayent d'apprendre à partir des situations d'échec. Ces trois classes correspondent, pour les deux premières, aux améliorations apportées à l'étape de simplification et pour la dernière au traitement des échecs. Nous dressons ci-dessous une liste non exhaustive de quelques méthodes pour chacune de ces classes d'heuristiques.

Heuristiques « syntaxiques » Les heuristiques de branchements syntaxiques peuvent être vues comme des algorithmes gloutons dont le but est de sélectionner les variables qui, une fois affectées, génèrent le plus de propagations possibles ou permettent de satisfaire le plus de clauses. Toutes ces heuristiques sont basées sur l'utilisation d'une fonction d'agrégation permettant d'estimer l'effet de l'affectation d'une variable libre. Parmi ces approches, citons :

– **BOHM :** Cette heuristique, proposée par Buro et Kleine-Büning (1992), consiste à choisir la variable qui, pour l'ordre lexicographique, maximise le vecteur de poids $(H_1(x), H_2(x), \ldots, H_n(x))$ avec $H_i(x)$ calculé de la manière suivante :

$$H_i(x) = \alpha \times max(h_i(x), h_i(\neg x)) + \beta \times min(h_i(x), h_i(\neg x)) \tag{3.1}$$

où $h_i(x)$ est le nombre de clauses de taille i contenant le littéral x. Les valeurs de α et β sont choisies de manière heuristique. Dans (Buro et Kleine-Büning 1992), les auteurs suggèrent de fixer $\alpha = 1$ et $\beta = 2$.

– **MOM :** L'heuristique MOM « *Maximum Occurences in Minimum Size Clauses* » proposée par Goldberg (1979) sélectionne la variable ayant le plus d'occurrences dans les clauses de plus petites tailles. Une variable x est choisie de manière à maximiser la fonction suivante :

$$(f^*(x) + f^*(\neg x)) \times 2^k + f^*(x) \times f^*(\neg x) \tag{3.2}$$

où $f^*(x)$ est le nombre d'occurrences du littéral x dans les clauses les plus courtes non satisfaites. La valeur de k, tout comme le fait de décider qu'une clause est courte, est donnée de manière heuristique.

Cette heuristique a été améliorée à plusieurs reprises, en essayant par exemple d'équilibrer les arbres produits par l'affectation d'une variable (Dubois *et al.* 1996, Dubois et Boufkhad 1996, Freeman 1995, Pretolani 1996).

– **JW :** L'heuristique de branchement proposée par Jeroslow et Wang (1990) est basée sur le même principe que l'heuristique MOM. Les auteurs introduisent deux heuristiques, lesquelles sont analysées dans (Hooker et Vinay 1994, Barth 1995), afin de fournir un poids aux variables. Le poids d'un littéral ℓ de Σ est calculé à l'aide de la fonction suivante : $J(\ell) = \sum_{\alpha \in \Sigma | \ell \in \alpha} 2^{-|\alpha|}$.

La première heuristique (JW-OS) proposée consiste à sélectionner le littéral ℓ qui maximise la fonction $J(\ell)$ et la seconde (JW-TS) consiste à identifier la variable x qui maximise $J(x)+J(\neg x)$, et à assigner la variable x à vrai, si $J(x) \geq J(\neg x)$, et de l'assigner à faux sinon.

Heuristiques de type « *look-ahead* » Les heuristiques de type « *look-ahead* » consistent à anticiper le résultat de l'affectation d'une variable non encore affectée à l'aide de méthodes de filtrages (telles que la propagation unitaire). En d'autres termes, ce type d'approche effectue une exploration de l'arbre de recherche en largeur, localement et temporairement. Dans le cas où une contradiction est détectée pendant la simplification d'une formule Σ par le littéral ℓ, le littéral $\neg\ell$ est propagé au niveau courant. De cette manière, le nombre de variables éligibles pour la prochaine variable de branchement est réduit. Nous présentons ici une liste sommaire d'heuristiques de branchements basées sur ce concept :

– **PU :** Les heuristiques basées sur la propagation unitaire (PU) (Pretolani 1993, Freeman 1995, Li et Anbulagan 1997) utilisent la puissance de la PU afin de sélectionner plus précisément la prochaine variable à instancier. Cette méthode permet, contrairement à l'heuristique MOM, de considérer les propagations unitaires produites en cascade. Pour chaque variable libre x de la formule, $\Sigma^*_{|x}$ et $\Sigma^*_{|\neg x}$ sont calculés, la variable choisie est celle pour laquelle $|\mathcal{V}_{\Sigma^*_{|x}}| + |\mathcal{V}_{\Sigma^*_{|\neg x}}|$ est minimal. Cette méthode est plus discriminante, mais aussi plus coûteuse que MOM. Pour pallier ce problème, les méthodes qui l'exploitent réduisent son utilisation à un sous-ensemble des variables non instanciées ou à la partie haute de l'arbre de recherche (Li et Anbulagan 1997).

– **BSH :** L'heuristique BSH (*Backbone Search Heuristic*) proposée par Dequen et Dubois (2004) et implémenté dans le solveur KCNFS est fondée sur la notion de *backbone* [15]. Plus précisément, l'heuristique proposée sélectionne les variables ayant le plus de chance d'appartenir au *backbone*. Pour cela, le score $score(x) = \text{BSH}(x) \times \text{BSH}(\neg x)$ de chaque variable x non assignée est calculé et la variable x qui maximise $score(x)$ est sélectionnée comme prochain point de choix. La valeur BSH d'un littéral ℓ est calculée à l'aide de l'Algorithme 3.3 tels que : $\mathcal{B}(\ell)$ représente l'ensemble $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ des clauses binaires tel que $\forall 1 \leq i \leq n$ le fait de falsifier α_i implique que ℓ est satisfait, i représente le compteur permettant de contrôler le nombre de récursions possibles et $bin(\ell)$ (respectivement $ter(\ell)$) donne le nombre de clauses binaires (respectivement ternaires) possédant une occurrence de ℓ.

15. Un littéral appartient au *backbone* d'une formule si et seulement s'il appartient à tous les modèles de la formule.

Étant très gourmande en ressources, cette heuristique ne peut pas être utilisée avec un nombre important de récursions. Ce constat implique que le réglage du compteur i est très important. Afin de régler cette valeur de manière dynamique, les auteurs proposent de faire varier celle-ci en fonction de la hauteur du nœud considéré dans l'arbre de recherche. En effet, puisque les choix effectués en haut de l'arbre ont un impact très important sur la profondeur moyenne, il semble cohérent que la valeur de i en haut de l'arbre soit plus élevée qu'en bas.

Algorithme 3.3 : BSH(Σ : CNF, i : entier, ℓ : littéral)

1 $i \longleftarrow i - 1$;
2 calculer $\mathcal{B}(\ell)$;
3 **si** $i = 0$ **alors retourner** $\sum_{(u \vee v) \in \mathcal{B}(\ell)} (2 \times bin(\neg u) + ter(\neg u)) \times (2 \times bin(\neg v) + ter(\neg v))$;
4 **retourner** $\sum_{(u \vee v) \in \mathcal{B}(\ell)} \mathsf{BSH}(\neg u) \times \mathsf{BSH}(\neg v)$;

Heuristiques de type « look-back » Les heuristiques de type « look-back » tirent parti des échecs pour revenir à des points de choix pertinents (*intelligent backtracking*) ou pour conserver une information clef provenant d'une longue phase de recherche et de déductions (enregistrement de *nogoods* ou pondération des variables). Une méthode basée sur ce concept, consiste à pondérer les variables apparaissant dans les derniers conflits. Cette approche permet, en particulier, de focaliser la recherche sur les parties difficiles du problème. Parmi les heuristiques s'appuyant sur ce principe citons :

- **pondération des clauses conflits :** cette approche, proposé par Brisoux *et al.* (1999), est issue du constat suivant : lorsqu'une clause a été montrée insatisfiable dans une branche particulière de l'arbre de recherche, il est important que cette information ne soit pas négligée par la suite. Effectivement, il peut s'avérer intéressant de chercher à retrouver cette inconsistance le plus rapidement possible dans les autres branches de l'arbre de recherche. Une manière de réaliser ceci est de donner une priorité plus importante aux littéraux apparaissant dans la clause menant à l'échec. Pour cela, à chaque fois qu'une variable propositionnelle conduit à une inconsistance, la clause ayant amenée au conflit voit sont poids augmenté d'une certaine valeur. Un des avantages de cette méthode est de pouvoir être combinée facilement avec d'autres heuristiques de branchement telles que MOM ou JW. En effet, les poids induit sur les clauses peuvent être facilement transférés aux variables ;
- **VSIDS :** l'heuristique VSIDS (« *Variable State Independent Decading Sum* »), proposée par Zhang *et al.* (2001), est l'heuristique de branchement la plus utilisée dans les implantations modernes de DPLL. Cette heuristique associe un compteur à chaque variable appelé *activité*. Lorsqu'un conflit survient, une analyse de conflits (cf. 3.1.5.1) permet de déterminer la raison du conflit et les variables en cause dans ce conflit voient leur activité augmentée. Au prochain branchement, la variable ayant la plus grande activité est alors choisie comme variable de décision. En effet, puisque celle-ci est la plus impliquée dans les conflits, elle est donc jugée fortement contrainte. De façon à être précis, il est important de signaler que l'activité des variables est divisée périodiquement par une certaine constante. Le fait de diminuer l'activité permet de privilégier les variables récemment intervenues dans les conflits.

3.1.4.4 Heuristiques de polarité

Bien qu'étant aussi importante que l'heuristique de choix de variables, l'heuristique de choix de polarité est souvent passée sous silence lors de la description des solveurs SAT. En effet, lorsqu'une

variable est choisie comme point de choix il est nécessaire de lui associer une certaine valeur de vérité. De par la complexité algorithmique (qui revient à trouver une solution au problème), ce choix est effectué de manière heuristique. Parmi les heuristiques de phase les plus connues citons :

- **false :** dans le solveur MINISAT (Eén et Sörenson 2004) (qui est à ce jour un des solveurs les plus utilisé) une heuristique de phase très simple a été proposée. Celle-ci consiste à toujours affecter la variable à la valeur fausse (choisir $\neg x$) ;
- **JW :** Jeroslow et Wang (1990), proposent une mesure w, prenant en compte des informations sur le problème (taille et nombre de clauses contenant la variable), pour selectionner la polarité du prochain point de choix (choisir x si $w(x) \geq w(\neg x)$, $\neg x$ sinon). La fonction permettant de donner un score à une variable proposée par les auteurs est celle décrite lors de la présentation de l'heuristique du même nom (cf. équation 3.1.4.3) ;
- **occurence** : l'heuristique *occurence* est un cas particulier de l'heuristique JW où la fonction permettant de mesurer le poids d'une variable x est fournie par le nombre d'occurrence $\#occ(x)$ de celle-ci dans le problème (choisir x si $\#occ(x) \geq \#occ(\neg x)$, $\neg x$ sinon). Cette heuristique a pour objectif de satisfaire le maximum de clauses ;
- ***progress saving* :** Pipatsrisawat et Darwiche (2007) observent, de manière expérimentale, que dans le cadre d'une approche utilisant le *backtracking* les solveurs effectuaient beaucoup de travail redondant. En effet, lorsqu'un retour-arrière est effectué, le travail accompli pour résoudre les sous-problèmes traversés avant d'atteindre le conflit est perdu. Pour éviter cela, les auteurs proposent de sauvegarder la dernière polarité obtenue pendant la recherche dans une interprétation complète notée $\mathcal{P}$. Ainsi, lorsqu'une nouvelle décision sera prise, la variable se verra attribuée la même polarité que précédemment (choisir x si $x \in \mathcal{P}$, $\neg x$ sinon). De cette manière, l'effort pour satisfaire un sous-problème déjà résolu auparavant sera moins important. Le principal désavantage de cette approche est de ne pas assez diversifier la recherche. Afin de pallier ce problème, Biere (2009) propose d'« oublier » une partie de l'interprétation $\mathcal{P}$.

3.1.4.5 Retour arrière non chronologique et apprentissage

Une des caractéristiques de l'algorithme DPLL est qu'il remonte l'arbre de recherche jusqu'au nœud de décision précédent l'échec lorsqu'un conflit est atteint. Le fait d'effectuer un retour arrière chronologique peut dans certaines conditions être très problématique. En effet, la cause de l'apparition de l'échec peut être due à un autre point de choix, décidé plus haut dans l'arbre de recherche. Afin d'illustrer cette situation, considérons l'exemple suivant :

Exemple 3.13. *Soit* $\Sigma = \{(a \vee b), (\neg a \vee b \vee c), (\neg b \vee c \vee d), (\neg b \vee c \vee \neg d), (\neg b \vee \neg c \vee d), (\neg b \vee \neg c \vee \neg d), \Omega\}$ *une* CNF *telle que* Ω *est satisfiable et* $\mathcal{V}_\Omega \cap \{a, b, c, d\} = \emptyset$. *Considérons qu'un ordre d'affectation des variables est fourni par la séquence suivante* $[a, x_1, x_2, \ldots, x_n, c, b, d]$ *où* $\mathcal{V}_\Omega = \{x_1, x_2, \ldots, x_n\}$. *Si la variable* a *est affectée à faux initialement, il est nécessaire de développer l'ensemble de l'arbre de recherche associé à* Ω *avant de pouvoir réfuter* $\neg a$. *Cette situation est illustrée dans la figure 3.7.*

Comme nous pouvons le voir sur l'exemple précédent, ne pas considérer la cause réelle de l'échec peut conduire à un phénomène nommé « *trashing* » et qui consiste à explorer de façon répétitive les mêmes sous-arbres. Afin d'y remédier un certain nombre d'approches tendent à analyser le conflit rencontré en prenant en compte les décisions à l'origine de celui-ci. Cette analyse à pour but d'extraire des informations de manière à effectuer un retour arrière non chronologique (*backjumping*) et ainsi éviter de redécouvrir les mêmes échecs.

Ce type d'approche est étudié et appliqué dans de nombreux domaines en intelligence artificielle (système de maintien de vérité ATMS (McAllester 1980, Stallman et Sussman 1977), les problèmes

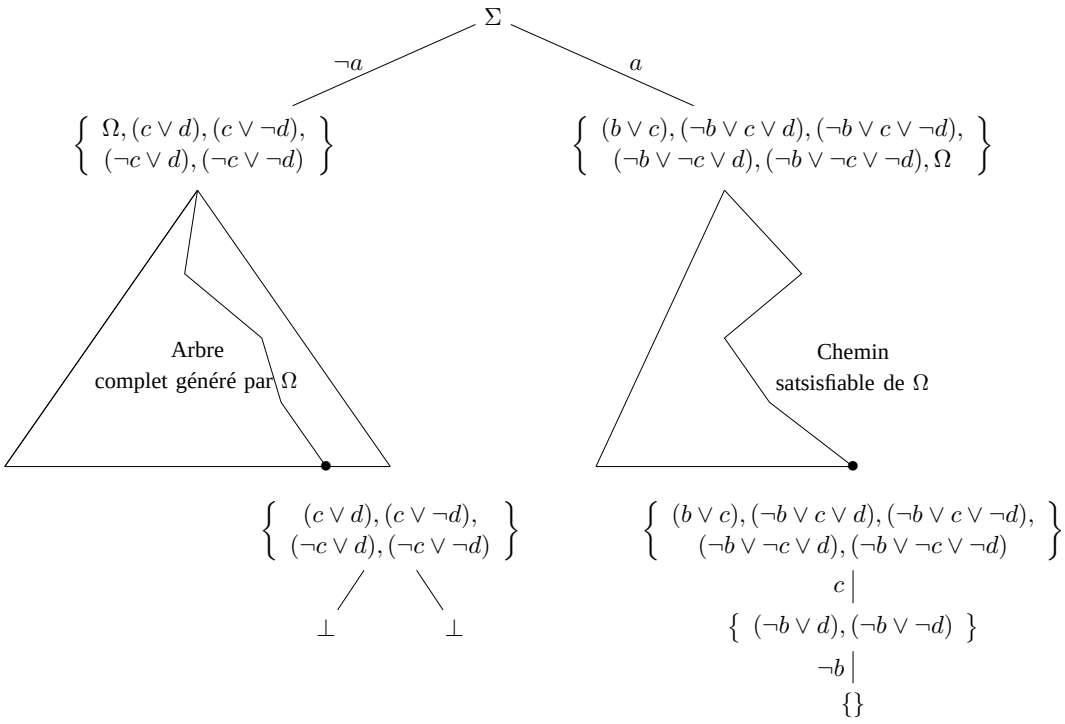


FIGURE 3.7 – Arbre de recherche généré par la procédure DPLL sur le problème de l'exemple 3.13.

de satisfaction de contraintes CSP (Dechter 1990a, Ginsberg 1993, Schiex et Verfaillie 1993), etc.). Ces techniques se distinguent essentiellement sur la manière dont les échecs sont analysés et les méthodes utilisées afin de ne plus se heurter aux mêmes situations par la suite. Dans le cadre de SAT ce n'est qu'en 1996 et par l'intermédiaire de Marques-Silva et Sakallah (1996) que l'analyse de conflits est introduite. Cette analyse de conflits permet d'extraire un terme $(x_1 \wedge x_2 \wedge \cdots \wedge x_n)$ signifiant que l'apparition du conflit est due à l'affectation conjointe des littéraux $x_1, x_2, \ldots, x_n$ à vrais (la manière d'extraire ces informations est présentée par la suite (voir §3.1.5.1)).

Malgré le fait d'effectuer un retour arrière non chronologique il est possible qu'une même situation d'échec se répète dans le futur. Afin d'éviter cela, une approche consiste à ajouter une information sous la forme d'une clause à la formule. Cette clause, nommée « nogood », est obtenue par la négation du terme construit par analyse de conflits $(\neg x_1 \vee \neg x_2 \ldots \neg x_n)$. L'ajout de cette clause, connue sous le nom de *clause apprise* (*clause learning*), permet à chaque fois qu'une contradiction est détectée d'identifier et d'ajouter une clause impliquée par la formule. De plus, considérer cette clause pour la suite de la recherche permet à la propagation unitaire de découvrir de nouvelles implications lesquelles permettent d'éviter de considérer la même situation d'échec plusieurs fois.

C'est la combinaison de ces principes et leurs intégrations au sein de l'algorithme DPLL qui ont conduit à l'élaboration des solveurs SAT modernes (Marques-Silva et Sakallah 1996, Zhang *et al.* 2001, Beame *et al.* 2004, Bayardo Jr. et Schrag 1997).

3.1.5 Solveur SAT moderne : CDCL

La procédure CDCL a été introduit par Marques-Silva et Sakallah (1996) (et améliorée par Moskewicz *et al.* (2001a)) et est une extension de la procédure DPLL. Cette méthode tente de tirer parti de l'analyse

de conflits afin d'apprendre une nouvelle clause et effectuer un retour arrière non chronologique. L'algorithme CDCL a deux avantages majeurs par rapport à l'algorithme DPLL : (i) effectuer un *backjumping* permet de ne pas considérer une partie de l'arbre de recherche ne possédant pas de solution et (ii) l'apprentissage de clauses permet d'éviter de considérer des sous-arbres de recherche, lesquels ont été montré inconsistants lors d'une précédente analyse de conflits. L'algorithme CDCL ne pouvant pas être présenté facilement de manière récursive, une version itérative de celui-ci est donnée dans l'algorithme 3.4.

Algorithme 3.4 : Solveur CDCL

Données : Σ une formule sous CNF
Résultat : SAT or UNSAT

```
1  Δ ⟵ ∅;                                   /* ensemble de clauses apprises */
2  I_p ⟵ ∅;                                 /* interprétation partielle */
3  dl ⟵ 0;                                  /* niveau de décision */
4  tant que (true) faire
5      α ⟵ propagationUnitaire(Σ ∪ Δ, I_p);
6      si (α = null) alors
7          x ⟵ heuristiqueDeBranchement(Σ ∪ Δ, I_p);
8          si (toutes les variables sont affectées) alors retourner SAT ;
9          ℓ ⟵affectePolarité(x);
10         dl ⟵ dl + 1;
11         I_p ⟵ I_p ∪ {(l^dl)};
12         si faireReduction() alors reductionClausesApprises(Δ);
13     sinon
14         β ⟵analyseConflit(Σ ∪ Δ, I_p, α);
15         bl ⟵calculRetourArrière(I_p, bl);
16         si β = ⊥ alors retourner UNSAT ;
17         Δ ⟵ Δ ∪ β;
18         retourArrière(Σ ∪ Δ, I_p, bl);          /* mise à jour de I_p */
19         si (faireRedémarrage()) alors redémarrage(I_p, dl);
```

Typiquement, l'algorithme CDCL peut être assimilé à une séquence de décisions suivie de propagations des littéraux unitaires, jusqu'à l'obtention d'un conflit. Chaque littéral choisi comme littéral de décision (ligne 7) est affecté suivant une certaine polarité (ligne 9) à un niveau de décision donné (ligne 11). Si tous les littéraux sont affectés, alors $\mathcal{I}_p$ est un modèle de Σ (ligne 8). À chaque fois qu'un conflit est atteint par propagation (lignes 13–19), une clause β est calculée en utilisant une méthode d'analyse de conflits donnée et un niveau de *backjump* est calculé en fonction de la clause β (lignes 14–15). À ce stade, il est possible de prouver l'inconsistance (β est la clause vide) de la formule (ligne 16). Si ce n'est pas le cas, un retour arrière est effectué et le niveau de décision devient égal au niveau de *backjump* (ligne 18). Ensuite, certains solveurs CDCL forcent le redémarrage et dans ce cas, un retour arrière est effectué au sommet de l'arbre de recherche (ligne 19). Finalement, la base de clauses apprises peut être réduite lorsque celle-ci est considérée comme trop volumineuse (ligne 12).

Afin de mieux juger des interactions entre les différentes composantes d'un solveur CDCL, voici ces éléments représentés sous la forme d'un diagramme (voir figure 3.8). Ce diagramme contient aussi une composante importante des solveurs SAT modernes qui est le prétraitement de la formule (voir §3.1.5.4).

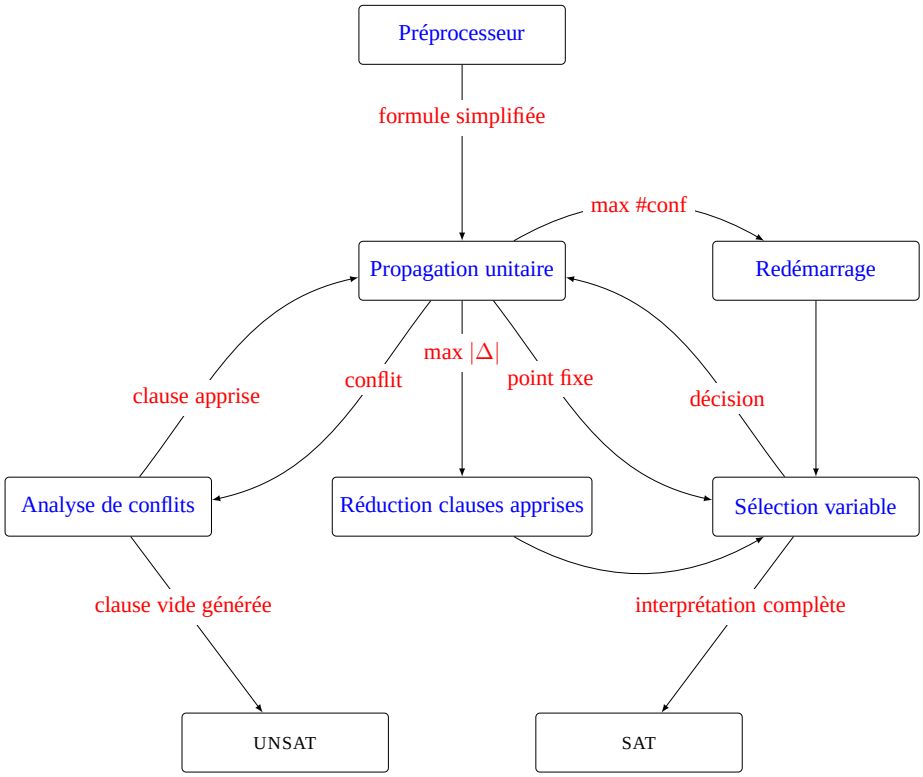


FIGURE 3.8 – Interaction des composantes d'un solveur CDCL.

Dans la suite, nous donnons une description des différentes fonctions laissées en suspens lors de la description.

3.1.5.1 Analyses de conflits basées sur l'analyse du graphe d'implications

Les premiers travaux sur l'analyse de conflits ont été réalisés sur les CSP (Prosser 1993). Les interactions entre le monde des CSP et celui de SAT ont permis l'élaboration et le développement de ces techniques dans le cadre de SAT (Marques-Silva et Sakallah 1996, Bayardo Jr. et Schrag 1997), qui, depuis, sont devenues incontournables.

Comme nous avons pu le souligner précédemment, le but de l'analyse de conflits est de trouver un ensemble de littéraux responsables d'une situation d'échec. Une fois cet ensemble de littéraux localisé une nouvelle clause est générée afin d'indiquer au solveur qu'il n'existe pas de solution dans un certain espace de recherche. À l'heure actuelle, le schéma d'analyse de conflits le plus couramment utilisé est basé sur l'analyse du graphe d'implications. Ce graphe est un graphe dirigé acylique (DAG) permettant de représenter les dépendances entre les clauses et les assignations obtenues par propagation des littéraux unitaires. Pour effectuer cela, un nœud est associé à chaque littéral affecté à vrai par l'interprétation courante. Ensuite, une arête entre un nœud x et un nœud y est créée si l'affectation de x à vrai a impliqué l'affectation de y à vrai ($x \in exp(y)$). Le graphe d'implications se définit formellement de la manière suivante :

Définition 3.15 (graphe d'implications)**.** *Soient Σ une formule et $\mathcal{I}$ une interprétation partielle. Le graphe d'implications associé à Σ et $\mathcal{I}$ est $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ tel que :*

- $\mathcal{N} = \{x \in \mathcal{I}\}$*, c'est-à-dire un nœud pour chaque littéral de $\mathcal{I}$, de décision ou propagé ;*
- $\mathcal{A} = \{(x, y)$ *tel que* $x \in \mathcal{I}, y \in \mathcal{I}$ *et* $x \in exp(y)\}$.

Exemple 3.14. *Soit la formule Σ suivante :*

$\alpha_1 = x_1 \vee x_2$ $\quad \alpha_4 = x_3 \vee x_5 \vee x_6$ $\quad \alpha_7 = \neg x_9 \vee x_{10} \vee x_{11}$ $\quad \alpha_{10} = x_{12} \vee x_{14}$
$\alpha_2 = \neg x_2 \vee \neg x_3$ $\quad \alpha_5 = \neg x_6 \vee x_7 \vee x_8$ $\quad \alpha_8 = x_8 \vee \neg x_{11} \vee \neg x_{12}$ $\quad \alpha_{11} = x_{12} \vee \neg x_6 \vee x_{15}$
$\alpha_3 = \neg x_2 \vee \neg x_4 \vee \neg x_5$ $\quad \alpha_6 = \neg x_4 \vee x_8 \vee x_9$ $\quad \alpha_9 = x_{12} \vee \neg x_{13}$ $\quad \alpha_{12} = x_{13} \vee \neg x_{14} \vee \neg x_{16}$
$\alpha_{13} = \neg x_{14} \vee x_{15} \vee x_{16}$

$\mathcal{I} = \{\langle(\neg x_1^1), x_2^1, \neg x_3^1\rangle, \langle(x_4^2), \neg x_5^2, x_6^2\rangle, \langle(\neg x_7^3), \neg x_8^3, \neg x_9^3\rangle, \langle(\neg x_{10}^4), x_{11}^4, \neg x_{12}^4, \neg x_{13}^4, x_{14}^4, x_{15}^4, x_{16}^4\rangle\}$ *est l'interprétation partielle obtenue par propagation de la séquence de décisions* $\langle \neg x_1, x_4, \neg x_7, x_{10}\rangle$. *Nous représentons en rouge (respectivement vert) les littéraux de la formule falsifiées (respectivement satisfait) par l'interprétation $\mathcal{I}$. L'interprétation $\mathcal{I}$ falsifie la formule (clause α_{13}). La figure 3.9 représente le graphe d'implications obtenu à partir de Σ et de l'interprétation $\mathcal{I}$.*

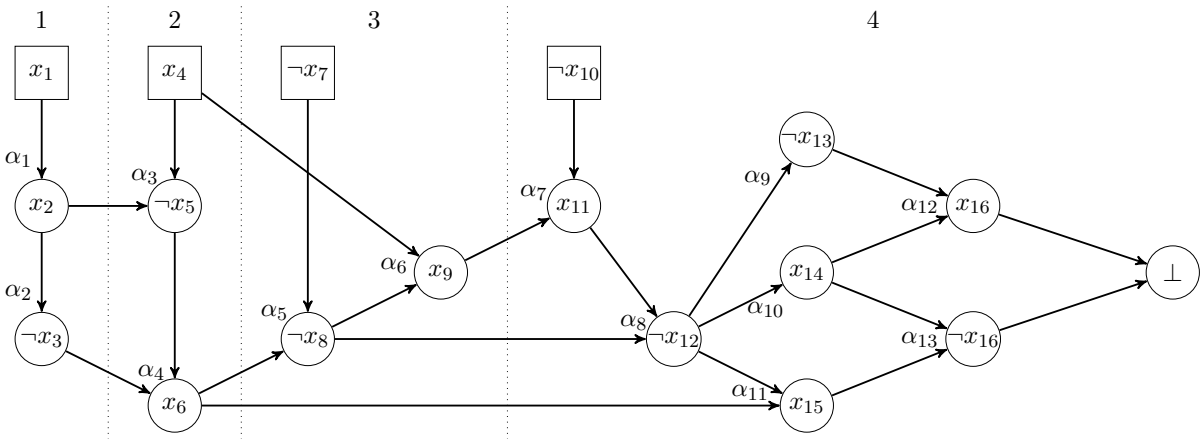


FIGURE 3.9 – Graphe d'implication obtenue à partir de la formule et l'interprétation partielle de l'exemple 3.14. Les nœuds de décision sont affichés en haut et sont représentés par des nœuds carrés. Les nœuds cercles représentent les littéraux affectés par propagation unitaire. Pour chacun d'entre eux, la clause responsable de cette affectation est annotée. Le conflit se trouve sur la variable x_{16}.

Lorsque l'interprétation partielle est conflictuelle, le graphe d'implications généré comporte deux nœud représentant deux littéraux complémentaires. Dans ce cas, il est possible d'ajouter un nœud supplémentaire ($\perp$) symbolisant une situation conflictuelle. Lorsqu'un tel nœud est présent il est possible d'analyser le graphe afin d'extraire les littéraux responsables de ce conflit. Généralement, cette analyse est basée sur la notion d'UIP (*Unique Implication Point*). Un UIP est un nœud du dernier niveau de décision qui domine le conflit. Formellement, nous avons :

Définition 3.16 (nœud dominant un niveau)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle et $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ le graphe d'implications généré à partir de Σ et $\mathcal{I}$. Un nœud $x \in \mathcal{N}$ domine un nœud $y \in \mathcal{N}$ si et seulement si $niv(x) = niv(y)$ et $\forall z \in \mathcal{N}$, avec $niv(x) = niv(z)$, tous les chemins de z vers y passent par x.*

Exemple 3.15. *Considérons le graphe d'implications associé à l'exemple 3.14. Le nœud $\neg x_{12}$ domine le nœud x_{16} tandis que le nœud x_{14} ne le domine pas.*

Définition 3.17 (point d'implication unique (UIP))**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle conflictuelle et $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ le graphe d'implication généré à partir de Σ en fonction de $\mathcal{I}$. Le nœud x est un point d'implication unique si et seulement si x domine le conflit.*

Remarque 3.5. *Les* UIP *peuvent être ordonnés en fonction de leur distance avec le conflit. Le premier point d'implication unique (*F-UIP *pour « First Unique Implication Point ») est l'*UIP *le plus proche du conflit tandis que le dernier* UIP *(*L-UIP *pour « Last Unique Implication Point ») est le plus éloigné, c'est-à-dire le littéral de décision affecté au niveau du conflit.*

Exemple 3.16. *Reprenons le graphe d'implications $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ généré dans l'exemple 3.14. Les nœuds $\neg x_{12}, x_{11}$ et $\neg x_{10}$ représentent respectivement le premier* UIP*, le second* UIP *et le dernier* UIP.

La localisation d'un UIP permet de fournir une décision alternative provoquant le même conflit. À l'heure actuelle, les démonstrateurs SAT modernes utilisent pour la plupart la notion de F-UIP afin d'extraire un *nogood* (Zhang *et al.* 2001). Cette clause est générée en effectuant des résolvantes entre les clauses responsables du conflit (utilisé durant la propagation unitaire) en remontant de celui-ci vers la variable de décision du dernier niveau jusqu'à obtenir une clause contenant un seul littéral du dernier niveau de décision (le UIP). Ce processus nommé preuve par résolution basée sur les conflits permet d'extraire une clause ne possédant qu'un seul littéral du dernier niveau. Cette clause est nommée clause assertive et est définie formellement comme suit :

Définition 3.18 (clause assertive)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation unitaire et m le niveau de décision courant. Une clause α de la forme $(\beta \vee x)$ est dite assertive si et seulement si $\mathcal{I}(\alpha) = \bot$, $niv(x) = m$ et $\forall y \in \beta, niv(y) < niv(x)$. Le littéral x est appelé littéral* assertif.

Définition 3.19 (preuve par résolution basée sur les conflits)**.** *Soient Σ une formule et $\mathcal{I}$ une interprétation partielle conflictuelle obtenue par propagation unitaire. Une preuve par résolution basée sur les conflits est une séquence de clauses $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ qui satisfait les conditions suivantes :*

- *$\sigma_1 = \eta[x, \overrightarrow{cla}(x), \overrightarrow{cla}(\neg x)]$, tel que $\{x, \neg x\} \subseteq \mathcal{I}$;*
- *$\forall 1 \leq i \leq k$, $\sigma_i = \eta[y, \sigma_{i-1}, \overrightarrow{cla}(\tilde{y})]$ telle que $y \in \sigma_{i-1}$;*
- *σ_k est une clause assertive.*

Il est important de souligner que dans le cadre des solveurs SAT modernes, les littéraux utilisés pour la résolution lors de la génération d'une preuve par résolution basée sur les conflits sont uniquement des littéraux du dernier niveau de décision. De plus, le littéral assertif obtenu à partir de la clause assertive générée par le biais de ce processus est un point d'implication unique (c'est ce processus qui est effectué par la fonction `analyseConflit`). Afin d'obtenir un F-UIP il est nécessaire que la preuve de résolution soit élémentaire, ce qui est défini formellement comme suit :

Définition 3.20 (preuve élémentaire)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle obtenue par propagation unitaire et $= \langle \sigma_1, \sigma_2, \ldots \sigma_n \rangle$ une séquence de résolution basée sur les conflits. La séquence de résolution est élémentaire si et seulement si $\nexists i < n$ tel que $\langle \sigma_1, \sigma_2, \ldots \sigma_i \rangle$ soit aussi une preuve par résolution basée sur les conflits.*

Exemple 3.17. *Considérons de nouveau la formule Σ et l'interprétation partielle $\mathcal{I}$ de l'exemple 3.14. La preuve par résolution basée sur les conflits est la suivante.*

$$\begin{array}{lcll} \sigma_1 & = & \eta[x_{16}, \alpha_{12}, \alpha_{13}] & = & x^4_{13} \vee \neg x^4_{14} \vee \neg x^4_{15} \\ \sigma_2 & = & \eta[x_{15}, \sigma_1, \alpha_{11}] & = & \neg x^2_6 \vee x^4_{12} \vee x^4_{13} \vee \neg x^4_{14} \\ \sigma_3 & = & \eta[x_{14}, \sigma_2, \alpha_9] & = & \neg x^2_6 \vee x^4_{12} \vee x^4_{13} \\ \sigma_4 & = & \eta[x_{13}, \sigma_3, \alpha_{10}] & = & \neg x^2_6 \vee x^4_{12} \\ \sigma_5 & = & \eta[x_{12}, \sigma_4, \alpha_8] & = & \neg x^2_6 \vee x^3_8 \vee \neg x^4_{11} \\ \sigma_6 & = & \eta[x_{11}, \sigma_5, \alpha_7] & = & \neg x^2_6 \vee x^3_8 \vee x^4_{10} \end{array}$$

Les littéraux assertifs des clauses assertives σ_4, σ_5 et σ_6 représentent respectivement le premier UIP*, le second* UIP *et le dernier* UIP.

La clause assertive correspondant au $1-$UIP ainsi générée est apprise par le solveur et permet non seulement d'éviter à l'avenir d'atteindre à nouveau cet échec, mais aussi d'effectuer un retour arrière. En effet, la connaissance de cette clause falsifiée permet d'affirmer que le littéral assertif doit être propagé plus tôt dans l'arbre de recherche. Ce calcul effectué à l'aide de la fonction `calculRetourArrière` est basé sur la propriété suivante :

Propriété 3.5 (clause assertive et retour arrière)**.** *Soient Σ une formule, $\mathcal{I}$ une interprétation partielle conflictuelle obtenue par propagation unitaire et $\alpha = (\beta \vee x)$ une clause assertive déduite à partir de l'analyse du conflit tel que $i = max\{niv(\tilde{y})$ tel que $y \in \beta\}$. Il est correct d'effectuer un retour arrière au niveau i et de propager le littéral x.*

Preuve 3.1. *La preuve de cette propriété est triviale et provient du fait que la clause α est unisatisfaite en x au niveau i.*

Exemple 3.18. *Considérons la clause assertive $\sigma = (\neg x_6^2 \vee x_{12}^4)$ généré à partir de la formule Σ et l'interprétation partielle $\mathcal{I}$ de l'exemple 3.14. Le niveau de backtrack calculé à partir de cette clause est égal à 2 et l'interprétation partielle obtenue après avoir effectué un retour arrière et avoir propagé le littéral assertif est $\mathcal{I} = \{\langle(\neg x_1^1), x_2^1, \neg x_3^1\rangle, \langle(x_4^2), \neg x_5^2, x_6^2, x_{12}^2\rangle\}$.*

Il est à noter qu'il est possible lors du processus d'analyse de conflits d'effectuer un travail supplémentaire. Par exemple, Sörensson et Biere (2009) proposent une approche qui consiste à continuer d'effectuer des résolutions si celles-ci n'augmentent pas la taille de la clause assertive. Plus récemment, deux travaux différents ont montré comment découvrir des clauses sous-sommées durant l'analyse de conflits (Han et Somenzi 2009, Hamadi *et al.* 2009a). Une autre approche, proposée par Audemard *et al.* (2008), tente d'étendre la notion de graphe d'implications afin de considérer certaines clauses satisfaites par la formule. Une dernière approche, proposée par Nadel et Ryvchin (2010), cherche à améliorer la hauteur du saut en effectuant un retour arrière à un niveau de décision inférieur à celui proposé par la clause assertive et à affecter et propager l'ensemble des littéraux de celle-ci.

Pour terminer, à chaque conflit les solveurs CDCL apprennent une nouvelle clause dont la taille peut être relativement importante. Dès lors, ils se heurtent à deux problèmes majeurs : l'utilisation de la mémoire et le temps utilisé pour la propagation unitaire. Pour remédier à cela, la base de clauses apprises est régulièrement réduite (Eén et Sörenson 2004, Goldberg et Novikov 2002). Un score, similaire au score de l'heuristique dynamique utilisée pour le choix des variables (VSIDS), est utilisé pour garder les clauses qui semblent importantes. Néanmoins, ce choix est heuristique et peut amener à supprimer des clauses essentielles pour la suite, ce qui peut s'avérer dramatique. En effet, nous savons que conserver un grand nombre de clauses peut avoir des conséquences néfastes sur l'efficacité de la propagation unitaire, mais en supprimer trop peut faire perdre le bénéfice de l'apprentissage. Par conséquent, identifier les bonnes clauses apprises (c'est-à-dire importantes pour la dérivation de la preuve) est un véritable challenge. Nous proposons dans le chapitre 9 une nouvelle mesure dynamique permettant d'évaluer la qualité des clauses apprises ainsi qu'un nouveau schéma de nettoyage de la base de clauses apprises.

3.1.5.2 Réduction de la base de clauses apprises

L'analyse de conflits et l'apprentissage ont permis d'améliorer de manière significative l'algorithme DPLL (Marques-Silva et Sakallah 1996). Cependant, comme le font justement remarquer Marques-Silva et Sakallah (1996), il est nécessaire de gérer l'accroissement de la base de clauses apprises au risque de ralentir considérablement le processus de propagation unitaire (Eén et Sörenson 2004). Pour éviter cela, tous les solveurs SAT modernes réduisent la base de clauses apprises (`reductionClausesApprises`)

à l'aide d'une fonction heuristique afin d'estimer la qualité des clauses. De plus, la plupart des approches conservent systématiquement les clauses de taille deux et les clauses considérées comme raison d'un littéral propagé au moment de l'appel à la fonction réduction. Pour ce qui est des clauses binaires, en plus de pouvoir être gérées facilement, leur conservation permet de contraindre fortement l'espace de recherche et donc d'améliorer la propagation unitaire. En ce qui concerne les clauses raisons ce n'est pas par soucis d'efficacité mais plus par nécessité qu'elles sont conservées. En effet, supprimer une clause apprise intervenant dans le processus de propagation unitaire ne permet plus d'assurer la construction du graphe d'implications et par conséquent l'analyse de conflits.

Dans la littérature diverses approches ont été proposés pour définir quelles sont les clauses susceptibles d'être inutiles pour la suite de la recherche. Parmi ces stratégies, deux ont montré des résultats pratiques intéressants. La première, la plus populaire, est basée sur la notion de *first fail*. Cette stratégie, basée sur l'heuristique VSIDS, considère qu'une clause apparaissant peu dans la raison des conflits est inutile et qu'elle doit être supprimée. La seconde est basée sur une mesure proposée par Audemard et Simon (2009b). Cette mesure appelé LBD (*Litetal Block Distance*), est une mesure statique qui correspond au nombre de niveaux différents intervenant dans la génération de la clause apprise. Nous détaillons ces diverses procédures ci-dessous.

VSIDS : Dans le solveur MINISAT (Eén et Sörenson 2004) une activité est associée à chaque clause apprise. Cette activité, calculée de la même manière que l'heuristique de choix de variable VSIDS, consiste à augmenter le poids des clauses touchées lors du processus d'analyse de conflits. Afin de donner une importance plus grande aux clauses récemment touchées, comme pour l'heuristique de choix de variable VSIDS, il est nécessaire de tenir compte de leur âge. Pour effectuer cela, le solveur gère deux variables : inc initialisée à 1 et $decay > 1$. Lorsqu'une clause est utilisée dans le processus d'analyse de conflits l'activité de celle-ci est augmentée de la valeur inc. Ensuite, après avoir mis à jour l'activité de l'ensemble des clauses touchées, la variable inc est augmentée telle que $inc = inc \times decay$. Lorsque la fonction de réduction est appelée l'ensemble des clauses apprises sont triées suivant cette activité et les clauses ayant les scores les plus faibles sont supprimées. En ce qui concerne la quantité de clauses à supprimer, les auteurs proposent d'effacer la moitié des clauses de la base.

LBD : La seconde heuristique est basée sur une mesure proposée par Audemard et Simon (2009b). Cette mesure est statique et correspond au nombre de niveaux différents intervenant dans la génération de la clause apprise. Formellement la valeur de LBD d'une clause se calcule comme suit.

Définition 3.21 (distance LBD (« *Literal Block Distance* »))**.** *Soient Σ une formule, α une clause et $\mathcal{I}$ une interprétation partielle associant un niveau d'affectation à chaque littéral de α. Alors la valeur de* LBD *de α est égal au nombre de niveaux de décision différents des littéraux de la clause α.*

Exemple 3.19. *Reprenons l'exemple 3.17, la clause assertive $(\neg x_6^2 \vee x_{12}^4)$ générée lors du processus d'analyse de conflits possède un* LBD *de deux.*

Dans (Audemard et Simon 2009b), les auteurs montrent expérimentalement que les clauses ayant une faible valeur de LBD sont importantes pour la suite de la recherche. Partant de ce constat, ils proposent d'utiliser cette mesure afin de donner un poids aux clauses apprises. Pour cela, lorsqu'une clause est générée par analyse de conflits son score est initialisée avec la valeur de LBD courante, c'est-à-dire calculée par rapport à l'interprétation courante. Ensuite, à chaque fois qu'elle est utilisée dans le processus de propagation unitaire son poids est ajusté dans le cas où la nouvelle valeur de LBD calculée vis-à-vis de l'interprétation courante est inférieure à la valeur actuelle. Comme pour le solveur MINISAT, lorsque

la fonction de réduction est appelé l'ensemble des clauses apprises sont triées suivant cette activité et la moitié des clauses ayant les scores les plus élevés sont supprimées.

Un autre point important concerne la fréquence avec laquelle la base de clauses apprises est nettoyée. Cette fréquence, calculée de manière heuristique (fonction `faireReduction`), et si elle est mal gérée, peut conduire à rendre le solveur incomplet. En effet, contrairement à la méthode DPLL, l'approche CDCL a besoin des clauses apprises afin de se souvenir de l'espace de recherche qu'elle a déjà exploré. Ceci implique que supprimer une clause peut amener le solveur à parcourir plusieurs fois le même espace de recherche et donc à visiter encore et encore le même conflit sans jamais s'arrêter. Afin de pallier ce problème et de garantir la complétude de la méthode, il est nécessaire que l'intervalle de temps atteigne une taille telle que l'ensemble des clauses apprises pouvant être généré dans cette intervalle permet d'assurer la terminaison de l'algorithme quelque soient les clauses supprimées précédemment. À l'heure actuelle deux approches sont principalement utilisées afin de gérer cette intervalle. La première approche, implémentée dans MINISAT (Eén et Sörenson 2004), consiste à appeler la fonction de réduction de la base de clauses apprises une fois que la taille de celle-ci a atteint un certain seuil. Ce seuil initialisé à $1/3$ de la taille de la base de clauses initiale est augmentée à chaque redémarrage de 10%. La deuxième approche est la stratégie implantée dans le solveur GLUCOSE (Audemard et Simon 2009a) vainqueur de la catégorie UNSAT de la compétition SAT en 2009. Elle consiste à réduire la base de clauses apprises lorsque le nombre de conflits obtenu depuis le dernier nettoyage est supérieur à $20000 + 500 \times x$ où x représente le nombre d'appels à la fonction de réduction.

3.1.5.3 Redémarrages

Comme nous l'avons fait remarquer lors de la présentation du phénomène de *trashing* (voir 3.1.4.5) les premiers choix effectués lors d'une recherche complète sont prépondérants. En effet Gomes *et al.* (2000) ont montré expérimentalement qu'exécuter la même approche sur le même problème mais avec des choix initiaux différents conduit à des temps de résolution totalement hétérogènes. Ces expérimentations ont permis d'identifier un phénomène singulier nommé phénomène de longue traînée (« heavy tail ») (Gent et Walsh 1994, Walsh 1999, Gomes *et al.* 2000, Chen *et al.* 2001). Afin d'y remédier les auteurs proposent de redémarrer la recherche au bout d'un certain temps (« *restart* »). L'idée est que si la recherche échoue depuis un certain nombre de temps (évalué en nombre de retour arrière) alors il est jugé peu probable que la recherche aboutisse en un temps raisonnable. Dans ce cas, l'algorithme est relancé avec la formule initiale tout en conservant certaines informations (les scores pour VSIDS par exemple) ou les clauses apprises afin d'effectuer des choix plus judicieux au début de l'arbre.

À l'heure actuelle le constat n'est plus exactement le même, l'utilité des redémarrages n'est plus reconnue comme étant d'essayer de chercher ailleurs dans l'espace de recherche, mais plutôt d'atteindre le même espace de recherche avec un chemin et des graphes d'implications différents (Biere 2008b).

Dans la suite nous présentons les stratégies de redémarrages les plus souvent utilisées (fonction `redémarrage`). Ces stratégies se décomposent en deux catégories. La première classe regroupe les stratégies de redémarrages statiques lesquelles sont établies au commencement de la recherche. La seconde classe regroupe les approches dynamiques qui tentent d'exploiter différentes informations en cours de recherche afin de déterminer si le solveur doit effectuer ou pas un redémarrage.

Stratégies de redémarrages statiques Les stratégies de redémarrages statiques ont toutes en commun le fait d'être prédéterminées. Ces politiques reposent le plus souvent sur des séries mathématiques. Voici quelques stratégies de redémarrages utilisées par différents solveurs :

– Intervalle fixe : Cette première stratégie somme toute très naïve consiste à appeler la fonction de redémarrage tous les x conflits. Elle est utilisée dans différents solveurs tels que SIEGE (Ryan 2004), ZCHAFF (Moskewicz *et al.* 2001b) (version 2004), BERKMIN (Goldberg et Novikov 2002) et EUREKA (Nadel *et al.* 2006) avec respectivement $x = 16000$, $x = 700$, $x = 550$ et $x = 2000$;
– Suite géométrique : Le solveur MINISAT 1.13 (Eén et Sörenson 2004) est le premier solveur à avoir démontré l'efficacité de la stratégie de redémarrage géométrique suggérée par Walsh (1999). Ce solveur commence par initialiser le premier intervalle de conflit autorisé à $limite = 100$. Ensuite, lorsque le nombre de conflit atteint la valeur $limite$, un redémarrage est effectuée et la valeur limite est augmentée de 50% ($limite = limite \times 1.5$) ;
– Luby : La série de Luby a été introduite dans (Luby *et al.* 1993) et a été utilisée pour la première comme stratégie de redémarrage dans le solveur TINISAT (Huang 2007). Cette stratégie évolue en fonction d'une série qui est de la forme : $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 4, 8, 1, 1, 2, \ldots$. Puisqu'il est très improbable de trouver une solution après un seul conflit, cette série est multipliée par un facteur. Formellement, l'intervalle entre deux redémarrages t_i et t_{i+1} est effectué après $u \times t_i$ conflits tels que u est une constante représentant le facteur multiplicateur et

$$t_i = \begin{cases} 2^{k-1} & \text{si } \exists k \in \mathbb{N} \text{ tel que } i = 2^k - 1 \\ t_{i-2^{k-1}+1}, & \text{sinon} \end{cases}$$

Cette stratégie a quelques caractéristiques théoriques intéressantes. En effet, les auteurs montrent dans (Luby *et al.* 1993) que cette approche est logarithmiquement optimale lorsqu'aucune information sur le problème n'est fournie. Ainsi à l'heure actuelle elle est devenue le choix des solveurs modernes tels que RSAT 2.0 (Pipatsrisawat et Darwiche 2007) et TINISAT (Huang 2007) qui l'utilise avec $u = 512$, ainsi que MINISAT 2.1 (Sörensson et Eén 2009) et PRECOSAT (Biere 2009) qui l'utilise avec $u = 100$;
– Inner-outer : Cette stratégie a été implémentée dans le solveur PICOSAT (Biere 2008b). Elle est similaire à la stratégie de redémarrage Luby (alternance de redémarrages rapides et lents) et consiste à maintenir une série géométrique extérieure ($outer$) dont le but est de fixer la borne maximale que peut atteindre une série géométrique intérieure ($inner$). Formellement, considérons i la valeur courante de la série intérieure et o la valeur courante de la série extérieure. Lorsque le nombre de conflits obtenu depuis le dernier redémarrage atteint la valeur i, un redémarrage est effectué et i prend la prochaine valeur de la série intérieure. Ensuite, dans le cas où $i > o$ la série intérieure est réinitialisée et o prend la prochaine valeur de la série extérieure.

L'impact de toutes ces différentes stratégies de redémarrages sur le comportement d'un solveur SAT moderne ont été étudiée dans (Huang 2007).

Stratégies de redémarrage dynamiques Depuis quelques temps, des stratégies de redémarrages dynamiques sont proposées. Elles tentent d'exploiter différentes informations issues de la recherche (la taille des résolvantes (Pipatsrisawat et Darwiche 2009b), la profondeur moyenne de l'arbre et des sauts arrière (Hamadi *et al.* 2010), le nombre récent de changements de valeur de variables Biere (2008a)) afin de déterminer quand effectuer un redémarrage. Nous présentons dans la suite une liste non exhaustive de ces différentes approches :

– Variation de la phase : Cette approche proposée par Biere (2008a) est l'une des premières politiques de redémarrages dynamique. L'idée est d'étudier l'agilité de la recherche afin de décider si un redémarrage est nécessaire. Cette agilité est calculée par l'analyse de l'interprétation complète représentant le *progress saving* (voir §3.1.4.4). L'agilité est initialisée à zéro ($agility = 0$). Ensuite, lorsque la polarité d'une variable est affectée sa nouvelle polarité est étudiée. Si la polarité de la variable est différente de l'ancienne alors l'agilité est augmentée, sinon elle est diminuée.

Afin de simuler une forme de « *aging* » sur l'affectation des variables, l'agilité est diminuée par un certain facteur[16] d compris entre 0 et 1 ($agility = agility \times d$). Ce même facteur est aussi utilisé afin d'augmenter l'agilité lors de la mise à jour de celle-ci ($agility = agility + (1 - d)$). De cette manière, il est assuré que la valeur de la variable $agility$ est comprise entre 0 et 1. La politique de redémarrage consiste alors à considérer une stratégie de redémarrage statique utilisant cette notion agilité. Lorsque le nombre de conflits à atteint la valeur préconisée par la politique de redémarrage statique, l'agilité courante est comparée avec un certain seuil[16] s et si elle dépasse ce seuil le redémarrage n'est pas effectué. Afin de régler ce seuil, Biere (2008a) émet l'hypothèse qu'une agilité élevée implique que le solveur est susceptible de réfuter le problème et donc qu'il ne faut pas effectuer de redémarrage. Et que inversement, une agilité basse implique une stagnation et redémarrer permet d'aider le solveur à s'échapper du sous arbre courant ;

- GLUCOSE : la plupart des politiques de redémarrages sont essentiellement basées sur le nombre de conflits afin de déterminer si un redémarrage doit être effectué. Audemard et Simon (2009a) proposent une nouvelle approche qui dépend non seulement du nombre de conflit, mais aussi des niveaux de décision afin de déterminer si un *restart* doit être effectué. Afin d'étudier les différents mécanismes des solveurs SAT modernes, les auteurs ont mené un ensemble d'expérimentations. Une des conclusions de celles-ci est que, sur une grande majorité d'instances, les niveaux de décisions décroissent tout au long de la recherche et que cette décroissance semble reliée avec l'efficacité (ou inefficacité) des solveurs CDCL (Audemard et Simon 2009b). Ainsi la politique de redémarrage proposée par les auteurs essaie de favoriser la décroissance des niveaux de décision durant la recherche : si ce n'est pas le cas, alors un redémarrage est effectué. Pour cela, le solveur GLUCOSE (Audemard et Simon 2009a) calcule la moyenne (glissante) des niveaux de décision sur les 100 derniers conflits. Si cette dernière est plus grande que 0.7 fois la moyenne de tous les niveaux de décision depuis le début de la recherche alors un redémarrage est effectué ;
- hauteur des sauts : Cette politique de redémarrage a été proposée par Hamadi *et al.* (2010) et a été implémentée dans le solveur LYSAT. Elle est basée sur l'évolution de la taille moyenne des retours arrière. L'idée est de délivrer pour de grandes (respectivement petites) fluctuations de la taille moyenne des retours arrière (entre le redémarrage courant et le précédant) une plus petite (respectivement grande) valeur de coupure. Cette fonction est calculée comme suit : $x_1 = 100, x_2 = 100$ et $x_i = y_i \times |cos(1 + r_i)|$, avec $i > 2$, $\alpha = 1200$, y_i représente la moyenne de la taille des retours arrière au redémarrage i, $r_i = \frac{y_{i-1}}{y_i}$ si $y_{i-1} > y_i$ et $r_i = \frac{y_i}{y_{i-1}}$ sinon.

3.1.5.4 Prétraitement de la formule

De manière générale le problème SAT est NP-complet (Cook 1971), c'est-à-dire que pour une formule comportant n variables tous les algorithmes connus à ce jour ont une complexité en $\mathcal{O}(2^n)$ dans le pire cas. En théorie, réduire le nombre de variables implique alors une diminution de la complexité dans le pire cas. Cependant, en pratique le nombre de variables n'est pas entièrement corrélé avec le temps de résolution. En effet, le nombre de clauses influence fortement les performances de la propagation unitaire, de l'apprentissage de nouvelles clauses et des retours arrière non chronologiques. Ainsi, réduire le nombre de variables tout en augmentant le nombre de clauses ne permet pas nécessairement d'accroître les performances des solveurs SAT mais au peut contraire avoir tendance à les réduire.

Les méthodes de prétraitement ont alors pour but de simplifier la formule par la réduction du nombre de variables et de clauses inutiles. Il existe différentes techniques lesquelles peuvent être séparées en deux groupes. Le premier regroupe les techniques qui conservent l'équivalence de la formule obtenue

16. $d = \frac{1}{10000}$ et $s = \frac{1}{4}$ pour l'analyse expérimentale

par simplification de la formule initiale. Le second regroupe les algorithmes qui visent à obtenir une formule qui n'est pas équivalente, mais équisatisfiable.

Des approches telles que la propagation unitaire, la subsumption ou la *self-subsumption* conservent l'équivalence (Heule *et al.* 2010) et ne sont pas décrites dans cette partie. Cependant, elles sont appliquées dans la plupart des algorithmes de prétraitement (Eén et Biere 2005) et permettent de supprimer des clauses et des littéraux redondants.

Dans la suite, nous décrivons quelques techniques de prétraitement. Les deux premières approches présentées ne préservent pas l'équivalence tandis que toutes les autres la conservent :

- **élimination de variables :** cette technique, implémentée dans SATELITE (Eén et Biere 2005, Subbarayan et Pradhan 2005), consiste à supprimer des variables de la formule initiale. L'élimination d'une variable est obtenue par l'application de la résolution sur toutes les clauses où cette variable apparaît. Cette technique, similaire à la procédure DP (voir §3.1.1), peut entraîner la création d'un nombre exponentiel de nouvelles clauses (exponentiel dans le nombre de variables de la formule). Afin de limiter cette croissance, les algorithmes de prétraitement n'appliquent cette opération que si la suppression d'une variable n'augmente pas la taille de la formule. Du fait de la suppression de variable, cette approche ne conserve pas l'équivalence de la formule. En effet, le modèle obtenu sur la formule prétraitée peut être partiel et ne pas satisfaire certaines clauses de la formule initiale. Néanmoins, Eén et Biere (2005) ont montré qu'il était possible d'étendre ce dernier en considérant les clauses de la formule initiale ;
- **élimination de clauses bloquées :** cette approche consiste à supprimer de la formule les clauses bloquées (Kullmann 1999). Une clause $\alpha \in \Sigma$ est dite bloquée si elle contient un littéral ℓ bloqué. Un littéral $\ell \in \alpha$ est bloqué si $\forall \alpha' \in \Sigma$ telle que $\neg\ell \in \alpha'$ la résolvante $\eta[\ell, \alpha, \alpha']$ est tautologique (Heule *et al.* 2010, Järvisalo *et al.* 2010). Les clauses bloquées sont des clauses redondantes et leur suppression est effectuée jusqu'à l'obtention d'un point fixe. Notons que l'ordre de suppression n'a aucune importance puisque la méthode est confluente (Järvisalo *et al.* 2010). De plus, puisque supprimer l'ensemble des clauses bloquées n'a pas de réelle influence sur les performances des solveurs, les méthodes de prétraitement n'utilisent pas les littéraux possédant de nombreuses occurrences. Comme pour l'élimination de variables, la suppression des clauses bloquées d'une formule ne permet pas de conserver l'équivalence (Heule *et al.* 2010) ;
- **élimination de clauses tautologiques dissimulées :** cette approche, proposée par Heule *et al.* (2010), est basée sur l'extension de clauses par ajout de littéraux, appelés littéraux dissimulés. L'ajout d'un littéral $\ell' \in \mathcal{L}_\Sigma$ à une clause $\alpha \in \Sigma$ est possible si $\exists \ell \in \alpha$ tel que $(\ell \vee \neg\ell') \in \Sigma \setminus \{\alpha\}$. Cette extension est appliquée jusqu'à l'obtention d'un point fixe. Les auteurs montrent que si la clause obtenue par l'ajout de tels littéraux sont tautologiques alors elles peuvent être supprimées de la formule initiale. Notons que l'application de cette technique permet d'obtenir une formule équivalente à la formule initiale (Heule *et al.* 2010) ;
- **élimination des équivalences :** le problème SAT étant exponentiel en fonction du nombre de variables de la formule, réduire leur nombre permet d'augmenter la vitesse de résolution. Pour effectuer cela, une méthode simple consiste à supprimer les littéraux équivalents et de conserver une représentation sous la forme d'une classe d'équivalence. De tels littéraux peuvent être détectés simplement par la recherche de cycles dans le graphe d'implications obtenu à l'aide de clauses binaires de la formule.
  Une autre manière de trouver de tels littéraux est d'utiliser la notion de *probing*. Le *probing* est une technique, proposée par Lynce et Marques-Silva (2003), permettant la simplification d'une formule par l'application de la propagation unitaire sur les deux polarités d'une même variable. Les auteurs proposent trois approches permettant d'extraire des informations des interprétations partielles ainsi obtenues. La première consiste à vérifier que l'application de l'une des deux po-

larités ne conduit pas à une interprétation incohérente, ce qui permet le cas échéant de propager l'un des deux littéraux. Afin d'illustrer les deux autres méthodes, considérons une formule Σ, une variable ℓ de Σ, $\mathcal{I}_\ell = (\Sigma_{|\ell})^*$ et $\mathcal{I}_{\neg\ell} = (\Sigma_{|\neg\ell})^*$ les deux interprétations partielles obtenues par l'application de la propagation unitaire. La seconde approche consiste alors à propager les littéraux qui apparaissent dans les deux interprétations partielles. En effet, supposons que $x \in \mathcal{I}_\ell \cap \mathcal{I}_{\neg\ell}$ alors $\Sigma \models (x \vee \ell)$ et $\Sigma \models (x \vee \neg\ell)$ [17] et donc $\Sigma \models x$. La dernière approche est celle qui nous intéresse puisqu'elle permet de détecter des équivalences. Pour cela, il suffit de vérifier s'il n'existe pas de littéraux apparaissant de manière complémentaire dans $\mathcal{I}_\ell$ et $\mathcal{I}_{\neg\ell}$. En effet, soit x tel que $x \in \mathcal{I}_\ell$ et $x \in \mathcal{I}_{\neg\ell}$ alors $(\ell \Rightarrow x)$ et $(\neg\ell \Rightarrow \neg x)$ et donc $(\ell \Leftrightarrow x)$;
vivification : cette dernière approche, proposée par Piette *et al.* (2008), consiste à réduire certaines clauses de la base par l'utilisation de la propagation unitaire. La réduction d'une clause $\alpha = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n)$ de Σ est obtenue en appliquant la propagation unitaire sur les littéraux complémentaires de α jusqu'à obtenir l'un des cas suivant :

1. $\Sigma_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models^* \bot$ avec $i < n$. Dans ce cas la clause α peut être remplacée par la clause $\alpha' = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_i)$. En effet, puisque l'interprétation $\{\neg\ell_1, \neg\ell_2, \ldots, \neg\ell_i\}$ falsifie Σ la clause α' peut être ajoutée à la formule initiale. Comme α' subsume α, il est possible de remplacer α par α' ;

2. $\Sigma_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models^* \ell_j$ tel que $\ell_j \in \alpha$ et $i < j < n$. Ici, le fait que $\Sigma_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models^* \ell_j$ nous permet d'inférer la clause $\alpha' = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_i \vee \ell_j)$ laquelle subsume α. De la même manière que pour le premier cas, la clause α peut être remplacée par α' ;

3. $\Sigma_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models^* \neg\ell_j$ tel que $\ell_j \in \alpha$ et $i < j \leq n$. Dans ce cas, $\Sigma_{|\{\neg\ell_1,\neg\ell_2,\ldots,\neg\ell_i\}} \models^* \neg\ell_j$ nous permet d'inférer la clause $\alpha' = (\ell_1 \vee \ell_2 \vee \cdots \vee \ell_i \vee \neg\ell_j)$. Puisque, α' self-subsume α il est possible de remplacer la clause α par $\eta[\ell_j, \alpha, \alpha']$.

Ces prétraitements font partie intégrante du processus de résolution pratique du problème SAT. En effet, à l'heure actuelle tous les solveurs effectuent ce genre de simplification avant de commencer la recherche de solutions. Ces techniques, basées sur l'analyse de la structure de la formule initiale, sont appliquées le plus souvent jusqu'à obtention d'un point fixe. Cependant, il est possible qu'en cours de recherche, des littéraux unitaires ou des clauses apprises modifient la base de connaissance de telle sorte que les prétraitements deviennent de nouveau effectifs. Ainsi, lorsqu'un littéral unitaire est appris ou que la recherche retourne au sommet de l'arbre (lors d'un redémarrage par exemple), toutes les techniques de simplifications peuvent de nouveau être appliquées (Eén et Biere 2005).

3.1.6 Synthèse

Nous avons abordé dans cette section la résolution pratique du problème SAT ainsi que la mise au point d'algorithmes performants pour le résoudre. Ces algorithmes de recherche reposent pour la plupart sur la procédure DPLL et sur une analyse à *posteriori* des échecs rencontrés. Cette analyse de conflits et l'apprentissage des *nogoods* qui en découle ont permis d'augmenter significativement la puissance des prouveurs SAT permettant ainsi de résoudre des problèmes de plus en plus conséquents. Ces avancées algorithmique bénéficient à d'autres problèmes NP-complets. Nous allons voir dans la section suivante que les approches complètes pour la résolution pratique du problème CSP sont très proches de celles utilisées dans le cadre de SAT.

17. Si $\Sigma_{|\ell} \models^* x$ alors $\Sigma_{|\ell\wedge\neg x} \models^* \bot$ et donc $\Sigma \models^* (\neg\ell \vee x)$

3.2 Approche complète pour la résolution pratique du problème CSP

Comme pour SAT, les principales méthodes de résolution complètes utilisées dans le cadre de CSP sont basées sur la combinaison d'un algorithme de type *backtracking* (Davis *et al.* 1962, Golomb et Baumert 1965) et d'une méthode de propagation de contraintes. Ces dernières sont basées sur la notion de cohérence locale et utilisent, comme pour la propagation unitaire (voir 3.1.4.1), des propriétés sur les contraintes afin de réduire l'espace de recherche de manière à supprimer les interprétations qui ne peuvent pas conduire à une solution.

Dans cette section, nous présentons l'adaptation des algorithmes de type *backtracking* au cadre CSP ainsi qu'une liste non exhaustive des techniques ayant permis d'accroître leur efficacité : schéma de branchements, méthode de cohérence locale, heuristique de choix de variables, heuristique de choix de valeurs, stratégie de randomisation et politique de redémarrages. Nous choisissons, puisqu'elles ne sont pas utiles pour la compréhension de la suite de ce manuscrit, de ne pas présenter l'analyse de conflits et les retours-arrière non chronologiques. Néanmoins, le lecteur intéressé par ces sujets peut se référer à la thèse de Tabary (2007) où un chapitre complet y est consacré (chapitre 2).

3.2.1 Générer et tester

Une première méthode (assez naïve) pour la résolution d'une instance CSP est de générer toutes les interprétations possibles, c'est-à-dire toutes les combinaisons possibles de valeurs des variables et de tester si celles-ci sont des solutions du problème, c'est-à-dire si elles vérifient toutes les contraintes du réseau. Cette approche, connue sous le nom de *Generate-and-test*, est similaire à l'approche basée sur les arbres sémantiques utilisée dans le cadre de SAT (voir §3.1.2). Comme cette dernière, elle évalue successivement toutes les interprétations possibles et n'est donc pas envisageable en pratique. En effet, afin d'obtenir l'ensemble des solutions d'un problème simple tel que les 8-reines, cet algorithme génère et vérifie la validité de $8^8 = 16777216$ interprétations différentes. Néanmoins, comme pour SAT avec l'algorithme de Quine (1950), ce nombre peut largement être réduit par une analyse de la structure du problème. Ces méthodes, appelées algorithmes de recherche avec retours arrière, sont présentées dans la partie suivante.

3.2.2 Algorithme de recherche avec retours arrière

L'algorithme de recherche avec retours arrière ((S)BT pour *(Standard) BackTracking*) (Bitner et Reingold 1975) est sans nul doute l'approche complète la plus utilisée pour la résolution pratique d'instances CSP. Il consiste à étendre de manière incrémentale une interprétation partielle en considérant une à une les variables du problème. Lorsqu'une variable est sélectionnée, une valeur de son domaine est choisie et lui est assignée. Ensuite, l'ensemble des contraintes du réseau est vérifié afin de savoir si aucune d'entre elles n'est falsifiée par la nouvelle interprétation. Si aucune contrainte n'est violée, l'assignation de la variable est un succès et une nouvelle variable est considérée. Dans le cas contraire, une autre valeur pour la variable est sélectionnée et assignée. Lorsque toutes les variables ont été assignées à une valeur et qu'aucune contrainte du réseau n'est violée, le problème est résolu et l'interprétation courante représente un modèle. Si à un moment de la recherche une variable ne possède plus aucune valeur à assigner un retour arrière est effectué. Dans ce cas, la dernière variable considérée est remise en cause et une nouvelle valeur lui est assignée si cela est possible, sinon un retour arrière est de nouveau effectué jusqu'à l'obtention d'une interprétation consistante. Cette procédure est effectuée tant qu'une solution n'a été

trouvée ou que l'absence de solution n'a pas été démontrée, c'est-à-dire tant que l'ensemble des interprétations n'a pas été parcourue. Puisqu'il permet d'élaguer certaines branches de l'arbre de recherche, l'algorithme de *backtracking* est strictement plus efficace que l'algorithme *Generate-and-test*.

Cet algorithme peut aussi être vu comme le parcours en profondeur d'abord d'un arbre de recherche où la racine correspond au réseau de contraintes initial et chaque nœud à un sous-réseau de contraintes (du réseau initial) à résoudre. La construction d'un tel arbre se fait généralement selon deux schémas de branchements différents : le schéma de branchement binaire et le schéma de branchement non-binaire. Chaque branche de l'arbre de recherche correspond à une *séquence de décisions*, c'est-à-dire des assignations et/ou des réfutations dans le cas d'un schéma de branchement binaire, et des assignations dans le cas d'un schéma de branchement non-binaire. Une solution du problème est obtenue lorsque le domaine de toutes les variables ont été réduit à des singletons et que toutes les contraintes sont satisfaites. Cette solution correspond à l'interprétation courante, c'est-à-dire aux assignations effectuées le long de la branche menant de la racine à une feuille de l'arbre de recherche. Lorsque l'arbre de recherche a été complètement exploré et qu'aucune solution n'a pu être identifiée alors le réseau de contraintes a été démontré insatisfaisable.

3.2.2.1 Schéma de branchement non-binaire

L'algorithme 3.5 décrit une recherche avec retour arrière où un schéma de branchement non-binaire est utilisé. Il consiste à sélectionner une nouvelle variable à l'aide de la fonction `HeuristiqueDeBranchement` (décrit dans le §3.2.3.3) et à tester successivement toutes les valeurs de son domaine (lignes 2-3), ce qui correspond à chaque fois à un nœud de l'arbre de recherche. Si l'assignation d'une valeur mène à un échec, elle est remise en cause et une autre valeur appartenant au domaine de X est assignée. Lorsque toutes les valeurs de la variable X ont été testées sans succès, l'algorithme effectue un retour-arrière et remet en cause le dernier choix de valeur pour la variable précédemment sélectionnée (ligne 6).

Algorithme 3.5 : BT-Non-Binaire

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes
Résultat : `vrai` si le réseau est satisfiable, `faux` sinon

```
1 Début
2     X ⟵ HeuristiqueDeBranchement(𝒳);
3     pour chaque v ∈ X faire
4         si 𝒫|(X=v) est cohérent alors
5             si (|𝒳| = 1) alors retourner vrai retourner BT-Non-Binaire(⟨𝒳 \ {X}, 𝒞⟩);
6     retourner faux;                    /* aucune solution sur cette branche */
7 Fin
```

La figure 3.10 illustre un arbre de recherche partiel correspondant au problème des 4-reines (décrit en section 1.3.1.2) construit par l'algorithme 3.5 où les variables et les valeurs sont ordonnées selon l'ordre lexicographique.

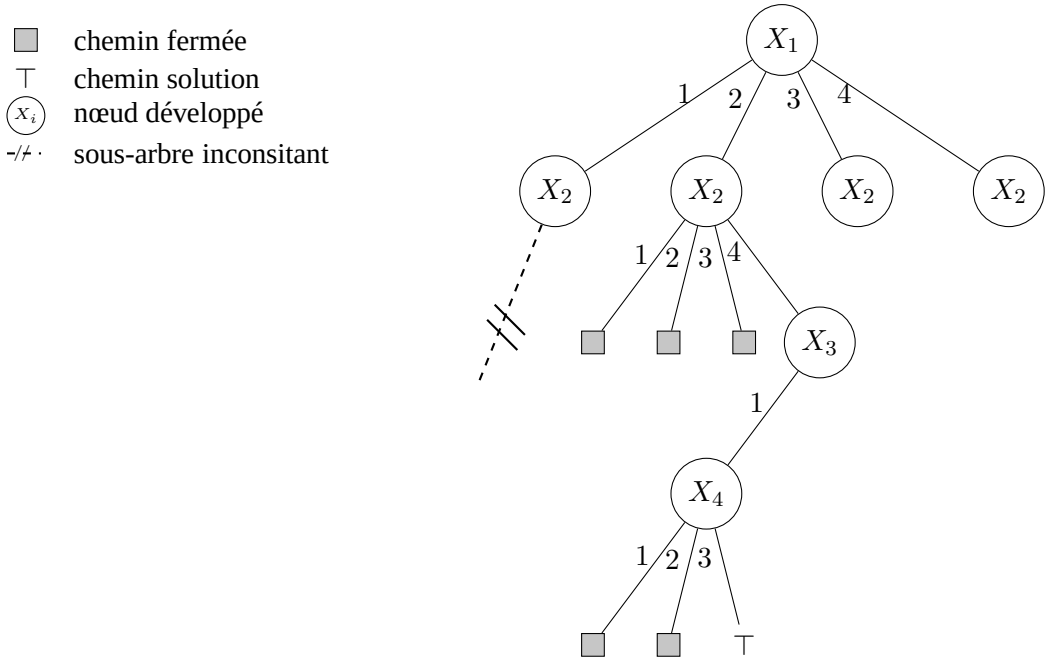


FIGURE 3.10 – Arbre de recherche partiel construit par un algorithme de recherche avec retours arrière utilisant un schéma de branchements non-binaires.

3.2.2.2 Schéma de branchement binaire

L'algorithme 3.6 décrit une recherche avec retours arrière où un schéma de branchement binaire est utilisé. Ce dernier consiste à sélectionner un couple (X, v) tel que X est une variable du réseau qui n'est pas encore assignée et v est une valeur de $dom(X)$ (ligne 4). Ensuite, l'algorithme construit les deux branches issues des décisions positives et négatives de (X, v).

Définition 3.22 (décisions positives et négatives)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $X \in \mathcal{X}$ et $v \in dom(X)$. $(X = v)$ (respectivement $(X \neq v)$) représente la décision positive (respectivement négative). Nous avons $\neg(X = v) = (X \neq v)$ et $\neg(X \neq v) = (X = v)$.*

Ces deux branches représentent les deux sous-problèmes obtenus en assignant ou en réfutant la valeur v du domaine de X (ligne 5). De manière générale, la branche positive est explorée en premier. Lorsque celle-ci est montrée incohérente (ligne 2), la branche négative est considérée. Dans le cas ou cette dernière conduit aussi à un réseau incohérent (ligne 2) un retour-arrière est effectué. Une solution du problème est obtenue lorsque toutes les variables sont singletons et que le problème est consistant (ligne 3).

Algorithme 3.6 : BT-Binaire

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes
Résultat : `vrai` si le réseau est satisfiable, `faux` sinon
1 **Début**
2 | **si** $\mathcal{P}$ *est incohérent* **alors retourner** `faux` ;
3 | **si** $\forall X \in \mathcal{X}, |dom(X)| = 1$ **alors retourner** `vrai` ;
4 | choisir un couple (X, v) tel que $X \in \mathcal{X}$ et $v \in dom(X)$;
5 | **retourner** `BT-Binaire(`$\mathcal{P}_{|(X=v)}$`)` ou `BT-Binaire(`$\mathcal{P}_{|(X\neq v)}$`)`;
6 **Fin**

La figure 3.11 illustre un arbre de recherche partiel obtenu par l'application de l'algorithme 3.6 sur le problème des 4-reines où l'ordre lexicographique est utilisé afin de sélectionner le prochain couple (variable, valeur).

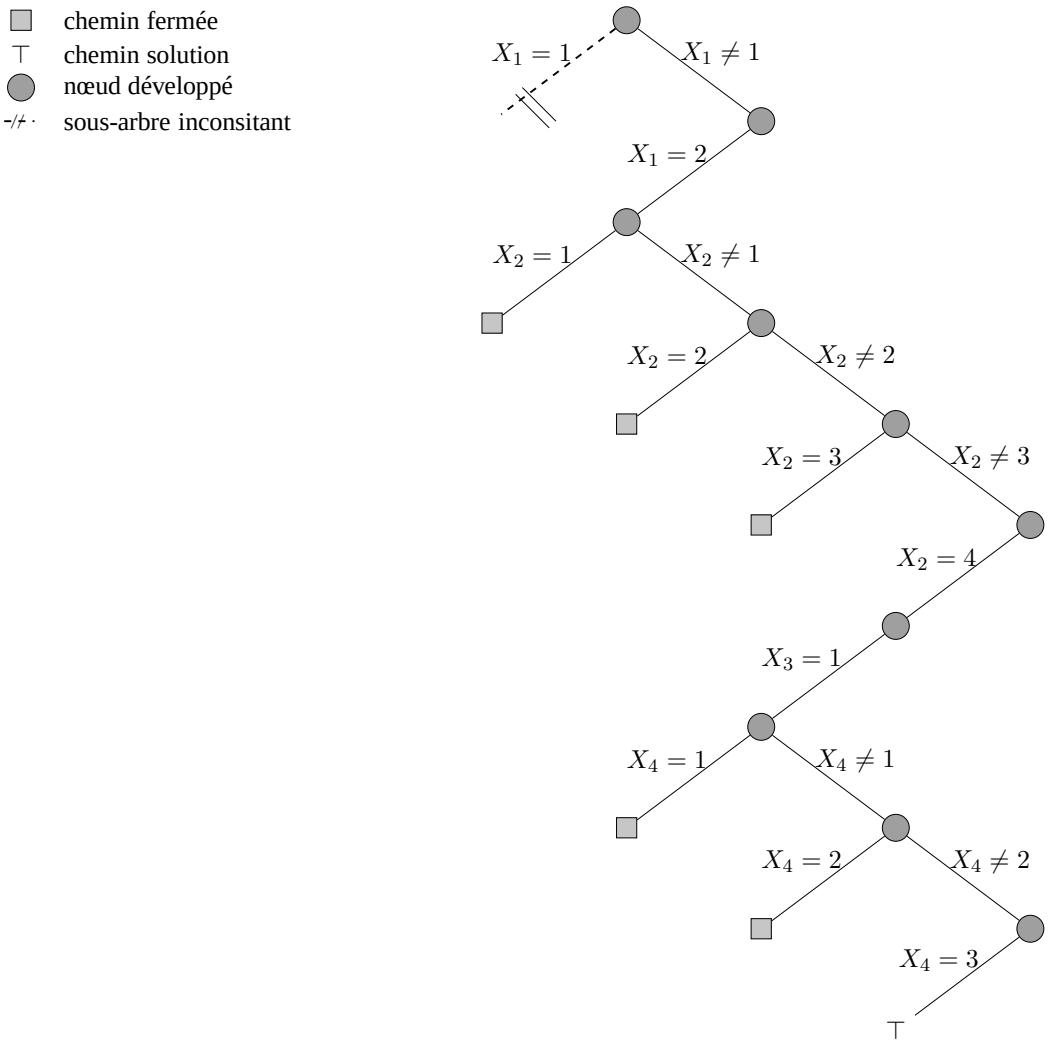


FIGURE 3.11 – Arbre de recherche partiel construit par un algorithme de recherche avec retours arrière utilisant un schéma de branchements binaires.

Bien que le schéma de branchements non-binaires puisse facilement être simulé par le branchement binaire, ces deux schémas de branchements ne sont pas équivalents. En effet, Hwang et Mitchell (2005) ont montré qu'il existe des instances pour lesquelles utiliser un schéma de branchements non-binaires est exponentiellement moins efficace que d'utiliser un schéma de branchements binaires. Dans la suite de ce manuscrit nous nous intéressons uniquement au schéma de branchements binaires.

L'algorithme de recherche avec retour arrière peut être sensiblement amélioré grâce à l'utilisation d'heuristiques de choix de variables et de valeurs efficaces (voir 3.2.3.3). Un autre levier permettant d'améliorer les performances d'une telle méthode est, comme pour SAT et l'algorithme DPLL, d'utiliser des méthodes de simplifications afin de détecter plus haut dans l'arbre de recherche que le chemin courant ne conduit à aucune solution. L'utilisation de ces améliorations conduit à la création d'algorithmes prospectifs que nous décrivons dans la partie suivante.

3.2.3 Algorithme de recherche avec retours arrière et filtrage

Afin d'améliorer les performances de l'algorithme de *backtracking*, il est possible d'exécuter un algorithme de filtrage afin de déterminer le plus tôt possible si l'interprétation courante peut conduire à une solution. Pour ce faire, les algorithmes complets de résolution vont appliquer un processus de simplification à chaque nœud de l'arbre de recherche. Ces méthodes, basées le plus souvent sur des cohérences locales (voir 3.2.3.1), permettent d'identifier et de supprimer les valeurs qui ne peuvent pas faire partie d'une solution.

L'algorithme 3.7 décrit une méthode de type *backtracking* où une fonction de filtrage ϕ est utilisée à chaque nœud de l'arbre. La fonction ϕ s'applique sur un réseau de contraintes et retourne un autre réseau de contraintes. Lorsqu'aucune forme de filtrage n'est effectuée, c'est-à-dire que ϕ retourne le même réseau, l'algorithme 3.7 est identique à l'algorithme 3.6.

Algorithme 3.7 : BT-filtrage

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, ϕ une méthode de filtrage
Résultat : `vrai` si le réseau est satisfiable, `faux` sinon

1 **Début**
2 $\quad \mathcal{P}' = \langle \mathcal{X}', \mathcal{C} \rangle \longleftarrow \phi(\mathcal{P})$;
3 $\quad$ **si** $\mathcal{P}'$ *est incohérent* **alors retourner** `faux` ;
4 $\quad$ **si** $\forall X \in \mathcal{X}', |dom(X)| = 1$ **alors retourner** `vrai` ;
5 $\quad$ choisir un couple (X, v) tel que $X \in \mathcal{X}'$ et $v \in dom(X)$;
6 $\quad$ **retourner** `BT-filtrage(`$\mathcal{P}'_{|(X=v)}$`)` ou `BT-filtrage(`$\mathcal{P}'_{|(X\neq v)}$`)`;
7 **Fin**

Plusieurs formes de cohérences locales peuvent être utilisées en fonction des caractéristiques des contraintes. Dans la section suivante nous présentons quelques schémas de cohérence.

3.2.3.1 Cohérence locale

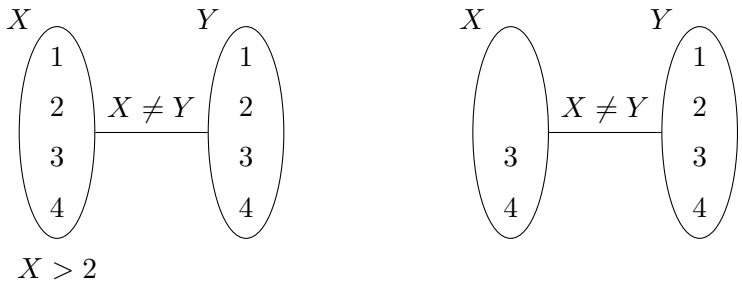
L'idée d'incorporer une méthode de propagation de contraintes au sein d'un algorithme de *backtracking* est apparue initialement dans le cadre de SAT et de l'algorithme DPLL (Davis et Putnam 1960). Ces méthodes sont, dans le cadre CSP, le plus souvent basées sur des formes de cohérences locales et permettent de vérifier qu'une instanciation partielle est localement cohérente. Pour cela, elles transforment le réseau de contraintes courant en un réseau de contraintes équivalent mais où les valeurs ne pouvant conduire à une solution sont supprimées. Dans la suite nous présentons un aperçu des formes les plus célèbres de cohérences locales.

Cohérence de nœud Cette forme de cohérence, appelée 1-cohérence, concerne les contraintes unaires et permet de vérifier que toutes les valeurs des variables sous la portée d'une contrainte unaire sont cohérentes. Il s'agit de la forme de cohérence la plus simple, puisqu'elle ne concerne qu'une variable.

Définition 3.23 (cohérence de nœud)**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, C une contrainte de $\mathcal{C}$ telle que $var(C) = \{X\}$, nous avons que :*

- *$v \in dom(X)$ est 1-cohérente si et seulement si $(v) \in \mathfrak{T}_a(C)$;*
- *X est 1-cohérente si et seulement si $\forall v \in dom(X)$, v est 1-cohérente ;*
- *$\mathcal{P}$ est 1-cohérent si et seulement si $\forall X \in \mathcal{X}$, X est 1-cohérente.*

Exemple 3.20. *Considérons le réseau de contraintes $\mathcal{P}$ de la figure 3.12. L'application de la cohérence de nœud sur $\mathcal{P}$ permet de supprimer les valeurs 1, 2 et 3 de la variable X. Le réseau de contraintes obtenu après ces suppressions est 1-cohérent.*



(a) réseau initial (b) réseau 1-cohérent

FIGURE 3.12 – Suppression de valeurs par cohérence de nœud.

Afin d'établir cette forme de cohérence il suffit de réduire le domaine de chaque variable aux valeurs qui satisfont les contraintes unaires de cette variable. Après avoir rendu le réseau de contraintes 1-cohérent, les contraintes unaires peuvent être supprimées du réseau.

Cohérence d'arc La *cohérence d'arc* est sans aucun doute la méthode de propagation de contraintes la plus étudiée. Cette approche permet de garantir que chaque valeur du domaine d'une variable est consistante avec toutes les contraintes du réseau. En effet, comme nous pouvons le voir sur l'exemple suivant il n'est pas toujours possible de trouver un support pour toutes les valeurs du domaine d'une variable.

Exemple 3.21. *Considérons la micro-structure de la contrainte C définit dans la figure 3.13. Comme nous pouvons le remarquer, il est impossible pour la valeur 3 de la variable X de trouver une valeur v dans le domaine de Y telle que $(3, v)$ satisfasse la contrainte. Il est donc possible de supprimer 3 du domaine de X puisque cette valeur conduit irrémédiablement à falsifier C. De la même manière, la valeur 1 du domaine de la variable Y peut aussi être supprimée.*

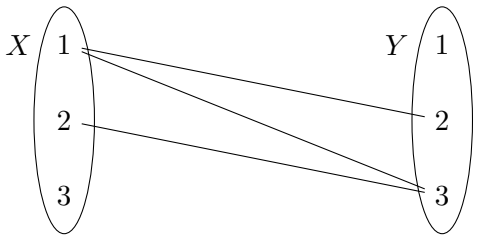


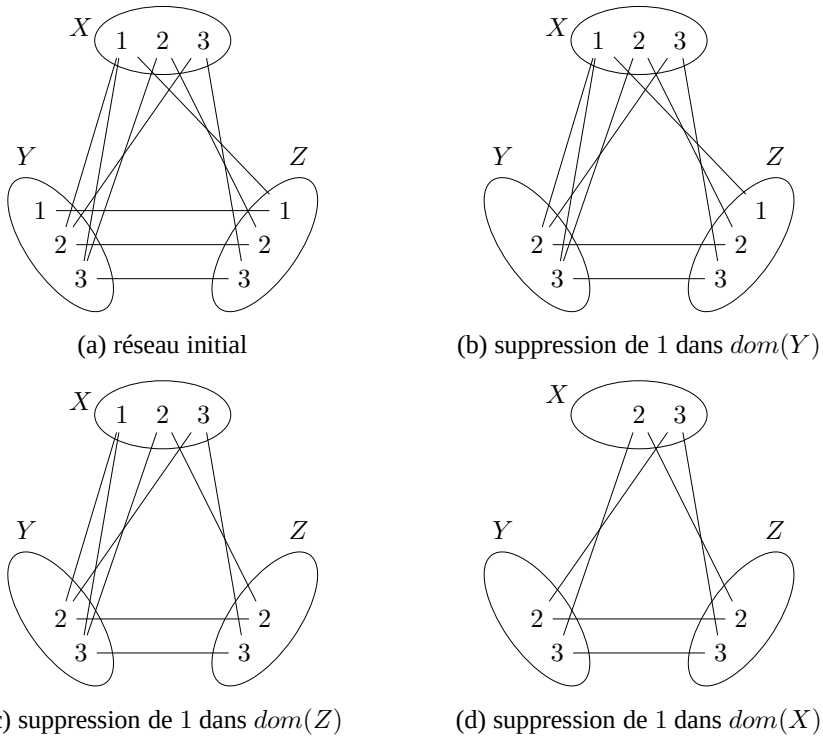
FIGURE 3.13 – Micro-structure des tuples autorisés d'une contrainte non arc cohérente.

Les premiers algorithmes permettant d'établir cette forme de cohérence ont été proposés par Waltz (1972) et Gaschnig (1974). Néanmoins, ce concept est introduit formellement par Mackworth (1977) dans le cas d'un réseau de contraintes binaires, il est par la suite étendu au cas de réseau n-aire (Mackworth 1977) et une analyse de sa complexité est effectuée (Mackworth et Freuder 1985). Nous donnons une définition de la cohérence d'arc dans le cas général, c'est-à-dire pour un réseau de contraintes quelconque (dans ce cas elle est souvent appelée cohérence d'arc généralisée GAC).

Définition 3.24 (Cohérence d'arc (généralisée) ((G)AC))**.** *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, C une contrainte de $\mathcal{C}$ telle que $var(C) = \{X_1, X_2, \ldots, X_n\}$, nous avons que :*

- *$v_i \in dom(X_i)$ est arc cohérente pour C si et seulement s'il existe un tuple $(v_1, v_2, \ldots, v_n) \in \mathfrak{T}_a(C)$ tel que $\forall j, 1 \leq j \leq n$, $v_j \in dom(X_j)$;*
- *X_i est arc cohérente (généralisée) pour C si et seulement si toutes les valeurs de $dom(X_i)$ sont arc cohérentes (généralisées) avec C ;*
- *$\mathcal{P}$ est arc cohérent (généralisé) si et seulement si $\forall X \in \mathcal{X}, \forall C \in \mathcal{C}$ nous avons X est arc cohérent (généralisée) pour C.*

Exemple 3.22. *Considérons la figure 3.14, le réseau de contraintes (a) reporté sur cette figure n'est pas arc cohérent. En effet, la valeur 1 du domaine de la variable Y ne possède pas de support pour la contrainte entre X et Y. Cette valeur peut donc être supprimée de Y. Cependant, cette suppression ne permet pas d'obtenir un réseau de contraintes arc cohérent puisque comme le montre la figure (b) la valeur 1 de Z ne possède pas de support pour la contrainte entre Y et Z. En fait, afin d'obtenir un réseau arc cohérent il faut supprimer la valeur 1 du domaine de Z (réseau (c)) et ensuite de supprimer la valeur 1 de X conduisant ainsi au réseau arc cohérent (d).*



(a) réseau initial

(b) suppression de 1 dans $dom(Y)$

(c) suppression de 1 dans $dom(Z)$

(d) suppression de 1 dans $dom(X)$

FIGURE 3.14 – Suppression de valeurs par cohérence d'arc.

Gaschnig (1974) est le premier à avoir suggéré un algorithme de *backtracking* avec maintien de la cohérence d'arc à chaque nœud de l'arbre de recherche. Cet algorithme, appelé DEEB (*Domain Element Elimination with Backtracking*), est basé sur un schéma de branchements non-binaires. Par la suite, Sabin et Freuder (1994) proposent une méthode avec maintien de la cohérence d'arc, appelée MAC (*Maintening (Generalized) Arc-Consistancy*), utilisant un schéma de branchements binaires. Ce dernier est à l'heure actuelle l'algorithme de résolution du problème CSP le plus efficace. La figure 3.15 illustre l'arbre de recherche obtenue par l'application de l'algorithme MAC où les heuristiques de choix de variables et de valeurs sont données par l'ordre lexicographique.

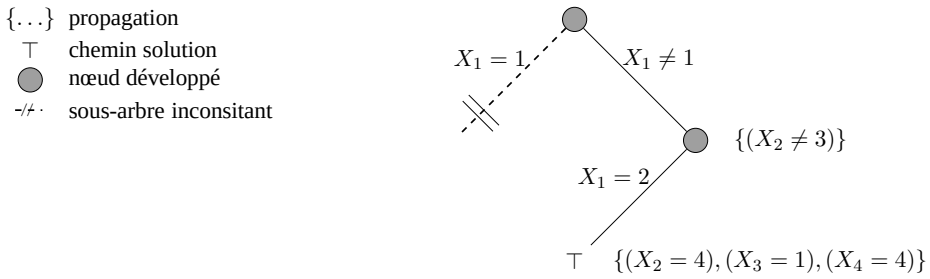


FIGURE 3.15 – Arbre de recherche partiel construit par l'algorithme MAC.

Comme le montre la figure 3.15, utiliser l'algorithme MAC permet de réduire fortement la taille de l'arbre. Néanmoins, pendant longtemps c'est une forme affaiblie de MAC, appelée *Forward Checking* (FC), qui obtenait les meilleures performances. Le *forward checking*, proposé par Haralick et Elliott (1980) dans le cas de réseaux binaires et par Bessière *et al.* (1999) dans le cas n-aire, établit contrairement à MAC une cohérence d'arc au voisinage des variables (et non pas à toutes les contraintes du réseau). Plus précisément, la propriété de cohérence d'arc est maintenue entre la variable assignée et les variables connectées à celle-ci par une contrainte.

La montée en puissance de l'algorithme MAC est en partie due aux développements d'algorithmes de maintien de la cohérence d'arc de plus en plus performants. En effet, comme le résume le tableau 3.1, les algorithmes proposés afin de maintenir AC à chaque nœud ont été largement étudiés. Chaque nouvelle version étant (en général) plus performante que la précédente. Ces algorithmes se distinguent selon qu'ils soient à gros ou à grain fin. Dans les approches à gros grain la suppression d'une valeur d'une variable X est propagée directement aux variables associées, c'est-à-dire que l'ensemble des variables connectées à X par une contrainte sont révisées. Dans un algorithme à grain fin, la suppression d'une valeur v de X implique uniquement la révision des valeurs des variables qui ont un lien avec v, c'est-à-dire les valeurs apparaissant dans un tuple autorisé par une contrainte C construit sur v. Bien qu'ils permettent d'effectuer moins de révisions inutiles les algorithmes à grain fin sont moins utilisés du fait de la difficulté à maintenir les structures de données nécessaires à leur mise en œuvre.

Algorithme	Temps	Espace	Grain	Auteur
AC3	$\mathcal{O}(e \times d^3)$	$\mathcal{O}(e)$	gros	Mackworth (1977)
AC4	$\mathcal{O}(e \times d^2)$	$\mathcal{O}(e \times d^2)$	fin	Mohr et Henderson (1986)
AC6 [18]	$\mathcal{O}(e \times d^2)$	$\mathcal{O}(e \times d)$	fin	Bessière et Cordier (1993)
AC7	$\mathcal{O}(e \times d^2)$	$\mathcal{O}(e \times d^2)$	fin	Bessière *et al.* (1999)
$AC3_d$	$\mathcal{O}(e \times d^3)$	$\mathcal{O}(e + n \times d)$	gros	Van Dongen (2006)
AC3.2/3.3	$\mathcal{O}(e \times d^2)$	$\mathcal{O}(e \times d)$	gros	Lecoutre *et al.* (2003)
AC2001/3.1	$\mathcal{O}(e \times d^2)$	$\mathcal{O}(e \times d)$	gros	Bessière *et al.* (2005)
$AC3^{rm}$	$\mathcal{O}(\frac{e \times d^2}{e \times d^3})$	$\mathcal{O}(e \times d)$	gros	Lecoutre et Hemery (2007)
AC2001-OP	$\mathcal{O}(e \times d^2)$	$\mathcal{O}(e \times d)$	gros	Arangú *et al.* (2010)

TABLE 3.1 – Algorithmes établissant la cohérence d'arc et leur complexité.

Certains de ces algorithmes, tel que AC2001/3.1, semblent plus attractifs du fait qu'ils possèdent une complexité temporelle optimale en $\mathcal{O}(ed^2)$ (Mohr et Masini 1988) tout en restant relativement simples à implémenter.

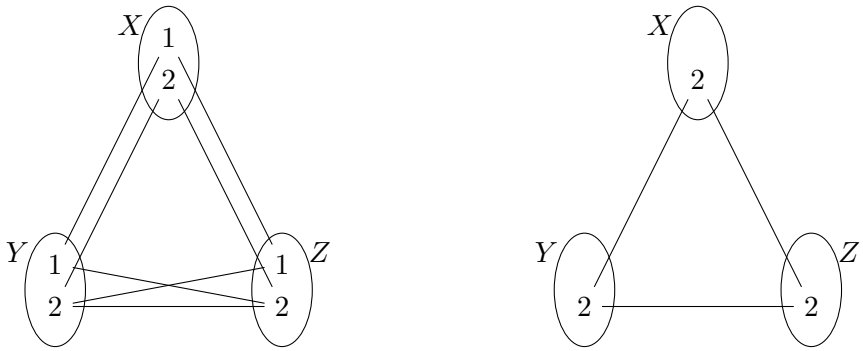
18. Cet algorithme utilise une structure de donné bien connue dans le monde de SAT : les *watched literals*

La consistance d'arc est sans nul doute la méthode de filtrage la plus utilisée à l'heure actuelle. Elle peut d'une certaine manière être rapprochée de la propagation unitaire utilisée dans le cadre de SAT. Néanmoins, plusieurs auteurs ont proposé à partir des années 70 d'autres techniques de filtrages beaucoup plus puissantes. Malgré leur puissance ces algorithmes sont pour la plupart uniquement appliquées comme méthodes de prétraitement. En effet, ces méthodes ont des complexités spatiales et temporelles trop élevées pour être maintenues à chaque nœud de l'arbre de recherche.

Cohérence d'arcs singletons (*Singleton Arc Cohérence* (SAC)) Cette forme de cohérence, proposée par Debruyne et Bessière (1997), consiste en quelque sorte à effectuer un parcours en largeur de profondeur 1 de l'arbre de recherche. Plus précisément, pour chaque variable X du réseau, les valeurs de X sont assignées une à une et la cohérence d'arcs est exécutée. Si le réseau obtenu après avoir effectué GAC est inconsistant, la valeur est supprimée.

Définition 3.25 (Cohérence d'arcs singletons)**.** *Un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ est cohérent d'arc singleton si et seulement si $\forall X \in \mathcal{X}$, $\forall v \in \mathcal{X}$, le sous-réseau $\mathcal{P}_{|(X=v)}$ est arc cohérent.*

Exemple 3.23. *Considérons le réseau de contraintes de la figure 3.16-(a). L'assignation de la valeur 1 à la variable X et l'application de la cohérence d'arcs rend le réseau inconsistant. La valeur 1 peut donc être supprimée de X. Cette suppression et l'application de* AC *implique les suppressions de 1 de Y et 1 de Z. Le réseau obtenu après ces suppressions, illustré dans la figure 3.16-(b), est* SAC.



(a) réseau initial (b) réseau SAC

FIGURE 3.16 – Suppression de valeurs par l'application de SAC.

De nombreux algorithmes ont été proposés afin d'établir la singleton arc-cohérence. Debruyne et Bessière (1997) proposent un algorithme « *brut force* » de complexité en temps de $\mathcal{O}(n^2 \times d^2 \times \mathcal{O}(\text{GAC}))$, nommé SAC1, qui consiste à tester une à une les interprétations de taille 1 et à relancer SAC chaque fois qu'une valeur est supprimée. Une autre approche, proposée par Barták et Erben (2004), reprend la philosophie de AC4 qui consiste à sauvegarder une liste de supports afin d'éviter du travail redondant. Bessiere et Debruyne (2005) proposent une approche, nommé SAC-OPT, de complexité temporelle en $\mathcal{O}(e \times n \times d^3)$ dans le cas de réseaux binaires. Cependant, afin d'être optimal, SAC-OPT utilise des structures de données conséquentes ($\mathcal{O}(e \times n \times d^2)$). Les auteurs proposent alors SAC-SDS une version non optimale mais plus « légère » de SAC-OPT qui effectue un compromis au niveau de la mémoire. Finalement, Lecoutre et Cardon (2005) proposent une approche, nommée SAC3, et qui contrairement aux autres approches ne redémarre pas à chaque suppression de valeurs.

Cohérence de chemin ((PC) *Path Consistency*) Cette forme de cohérence, proposée par Montanari (1974) dans le cas de réseaux binaires, consiste à vérifier, qu'étant donnés deux couples $\{(X, v), (X', v')\}$ et pour toute variable X'' voisine de X et X', il existe v'' dans le domaine de X'' telle que $\{(X, v), (X'', v'')\}$ et $\{(X', v'), (X'', v'')\}$ sont des interprétations partielles consistantes.

Définition 3.26 (cohérence de chemin)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, nous avons que :*

- *le chemin $((X, v), (X', v'))$, tels que $\{X, X'\} \subseteq \mathcal{X}$, $v \in dom(X)$ et $v' \in dom(X')$, est chemin-cohérent si et seulement si $\forall X'' \in \mathcal{X} \backslash \{X, X'\}$ il existe $v'' \in dom(X'')$ telle que $\{(X, v), (X'', v'')\}$ et $\{(X', v'), (X'', v'')\}$ sont des interprétations partielles consistantes ;*
- *$(X, X') \in \mathcal{X} \times \mathcal{X}$ est chemin-cohérent si et seulement si $\forall (v, v') \in dom(X) \times dom(X')$, $((X, v), (X', v'))$ est chemin-cohérent ;*
- *$\mathcal{P}$ est chemin-cohérent si et seulement si tout couple $(X, X') \in \mathcal{X} \times \mathcal{X}$ est chemin-cohérent.*

Exemple 3.24. *Considérons le réseau de contraintes $\mathcal{P}$ illustré dans la figure 3.17-(a). Ce réseau n'est pas chemin-cohérent puisque ni $((Y = 1), (Z = 2))$ ni $((Y = 2), (Z = 1))$ ne peuvent être étendus avec une valeur de X tout en satisfaisant les contraintes $(X \neq Y)$ et $(X \neq Y)$. Afin de rendre $\mathcal{P}$ chemin-cohérent, il suffit d'ajouter les tuples interdits $((Y = 1), (Z = 2))$ et $((Y = 2), (Z = 1))$ conduisant ainsi au réseau de contraintes de la figure 3.17-(b).*

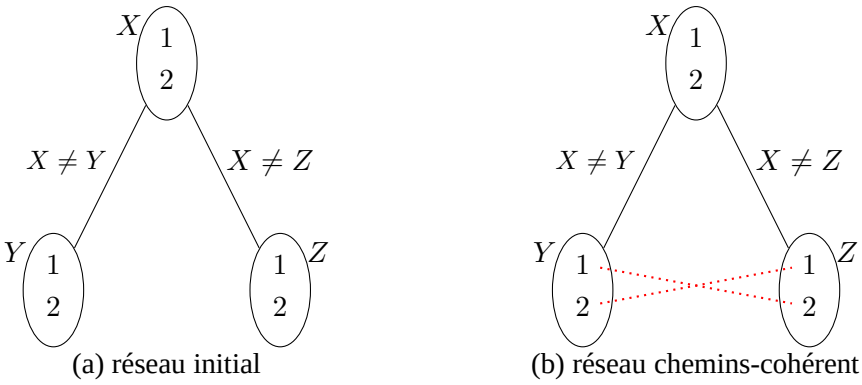


FIGURE 3.17 – Application de la cohérence de chemins.

Remarque 3.6. *Nous pouvons remarquer que la définition 3.26 ne s'applique que dans le cadre de contraintes binaires. En effet, les interprétations partielles considérées étant de taille deux, seules les contraintes binaires ont besoin d'être vérifiées.*

Contrairement à la cohérence d'arc, cette forme de cohérence implique de modifier les relations associées aux contraintes et même éventuellement la structure du graphe de contraintes associées au réseau. En effet, l'application de la chemin cohérence implique dans la plupart des cas de considérer un réseau de contraintes en extension.

Même si la consistance de chemin permet de filtrer plus de valeurs que la cohérence d'arc, il peut être coûteux de maintenir ce type de cohérence à chaque nœuds de l'arbre de recherche. En effet, la complexité en espace dans le pire des cas de cet algorithme est de $\mathcal{O}(n^2 d^2)$, correspondant au nombre d'interprétations de taille 2, et possède une complexité temporelle dans le pire des cas de $\mathcal{O}(n^4 d^5)$.

De nombreux algorithmes ont été proposés pour établir la cohérence de chemin d'un réseau. En fait, chaque fois qu'une nouvelle technique a été proposée pour la cohérence d'arc elle a été appliquée pour la consistance de chemin. PC1 (Montanari 1974, Mackworth 1977) peut être vue comme une extension de

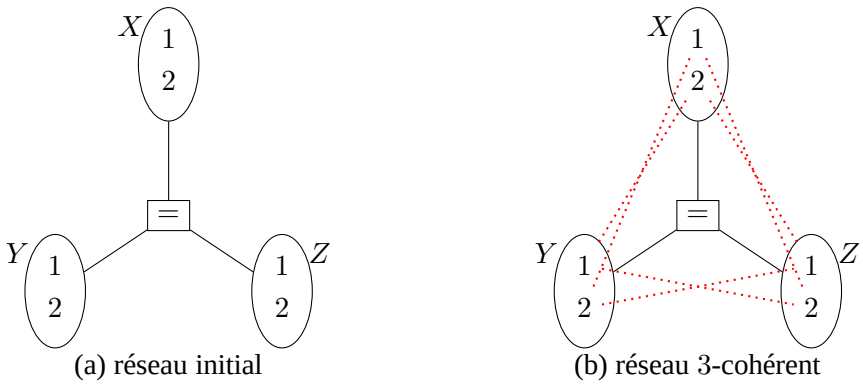
AC1, PC2 (Mackworth 1977) est une extension de AC3, PC3 (Mohr et Henderson 1986) et PC4 (Han et Lee 1988) utilisent une liste supports comme AC4, PC5 (Singh 1995) et PC6 (Chmeiss 1996) découlent de AC6, PC5++ (Singh 1995) applique la bidirectionnalité de AC7 et PC2001 (Zhang et Yap 2001, Bessière *et al.* 2005) est une extension de AC2001. Finalement, les algorithmes PC7 (Chmeiss et Jégou 1996) et PC8 (Chmeiss et Jégou 1998) sont quant à eux des amélioration de PC6.

La k-cohérence forte et la $(i-j)$-cohérence Peu de temps après le papier de Montanari (1974), Freuder (1978) a proposé une généralisation des cohérences d'arc et de chemin appelée k-cohérence.

Définition 3.27 (k-cohérence)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, nous avons que :*
- *$\mathcal{A} = \{(X_1, v_1), (X_2, v_2), \ldots, (X_{k-1}, v_{k-1})\}$ un ensemble d'assignations consistant, tel que $\forall i$, $X_i \in \mathcal{X}, v_i \in dom(X_i)$ et $X_i \neq X_j$ pour $1 \leq i < j \leq k-1$, est k-cohérent si et seulement si $\forall X' \in \mathcal{X} \setminus \{X_1, X_2, \ldots X_{k-1}\}$ il existe $v' \in dom(X')$ tel que $\mathcal{A} \cup \{(X', v')\}$ est consistant ;*
- *$\mathcal{Y} \subseteq \mathcal{X}$, de tel sorte que $|\mathcal{Y}| = k-1$, est k-cohérent si et seulement si quelque soit l'instanciation $\mathcal{A} = \{(X_1, v_1), (X_2, v_2), \ldots, (X_{k-1}, v_{k-1})\}$ consistante telle que $\forall i, X_i \in \mathcal{Y}$ telle que $\forall i, X_i \in \mathcal{Y}$, $\mathcal{A}$ est k-cohérente ;*
- *$\mathcal{P}$ est k-cohérent si et seulement si $\forall \mathcal{Y} \subseteq \mathcal{X}$ tel que $|\mathcal{Y}| = k-1$, $\mathcal{Y}$ est k-cohérent.*

La k-cohérence est une notion générale permettant de redéfinir les notions de cohérence présentées précédemment. En effet dans le cadre de réseau de contraintes binaires, la cohérence de nœud, d'arcs et de chemins correspondent respectivement à la 1-cohérence, 2-cohérence et 3-cohérence. Cependant, comme l'a démontrée Dechter (1990b), la cohérence de chemin n'est pas équivalente à la 3-cohérence dans le cadre n-aire. En effet, la cohérence de chemin garantit que tout couple de valeur est cohérent avec les contraintes binaires du réseau (voir remarque 3.6) tandis que la 3-cohérence permet de considérer des contraintes d'arité supérieures.

Exemple 3.25. *Considérons le réseau de contraintes $\mathcal{P}$ de la figure 3.18-(a). Puisque $\mathcal{P}$ ne possède pas de contraintes binaires, $\mathcal{P}$ est chemin cohérent. Cependant $\mathcal{P}$ n'est pas 3-cohérent. En effet, l'instanciation $\{(X_1 = 1), (X_2 = 2)\}$ ne peut pas être étendue à la variable X_3. Afin de rendre le réseau 3-cohérent il est nécessaire d'ajouter l'ensemble de tuples interdits représentés en pointillé sur la figure 3.18-(b).*



(a) réseau initial (b) réseau 3-cohérent

FIGURE 3.18 – Application de la 3-cohérence.

La k-cohérence permet d'assurer que toutes les instanciations de $k-1$ variables sont consistantes. La question est alors de savoir s'il est possible de construire une telle instanciation. Pour cela, la notion de k-cohérence forte a été introduite.

Définition 3.28. *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $\mathcal{P}$ est fortement k-cohérent si et seulement s'il est k'-cohérent pour tout $1 \leq k' \leq k$.*

Freuder (1985) propose une notion, appelée (i, j)-cohérence, permettant de généraliser la notion de k-cohérence. Cette approche emprunte le cheminement inverse qui consiste à vérifier s'il existe une instanciation cohérente de taille k en fixant une valeur d'une variable. De manière générale, est-il possible d'étendre i assignations à une instanciation de taille j.

Définition 3.29 ((i, j)-cohérence)**.** *Soit $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, nous avons que :*
- *$\mathcal{A} = \{(X_1, v_1), (X_2, v_2), \ldots, (X_i, v_i)\}$ un ensemble d'assignations consistant, tel que $\forall l, X_l \in \mathcal{X}, v_l \in dom(X_l)$ et $X_l \neq X_m$ pour $1 \leq l < m \leq i$, est (i, j)-cohérent si et seulement si $\forall \mathcal{Y} \subseteq \mathcal{X} \setminus \{X_1, X_2, \ldots X_i\}$ avec $|\mathcal{Y}| = j$, il est possible de construire une instanciation partielle $\mathcal{A} \cup \bigcup_{\substack{X \in \mathcal{Y} \\ v \in dom(X)}} (X, v)$ consistante ;*
- *$\mathcal{Y} \subseteq \mathcal{X}$, de telle sorte que $|\mathcal{Y}| = i$, est (i, j)-cohérent si et seulement si quelque soit l'instanciation $\mathcal{A} = \{(X_1, v_1), (X_2, v_2), \ldots, (X_i, v_i)\}$ consistante telle que $\forall i, X_i \in \mathcal{Y}$, $\mathcal{A}$ est (i, j)-cohérente ;*
- *$\mathcal{P}$ est (i, j)-cohérent si et seulement si $\forall \mathcal{Y} \subseteq \mathcal{X}$ tel que $|\mathcal{Y}| = i$, $\mathcal{Y}$ est (i, j)-cohérent.*

Remarque 3.7. *Comme nous l'avons souligné précédemment, il est possible de réécrire la k-cohérence comme la $(k-1, 1)$-cohérence et l'inverse k-cohérence comme la $(1, k-1)$-cohérence.*

Du fait de leur complexité élevée, les formes de cohérences locales au-delà de la 3-cohérence, forte ou non, sont rarement utilisées en pratique. En effet, afin d'effectuer la k-cohérence il est nécessaire d'énumérer dans le pire des cas les d^{k-1} instanciations de taille $k-1$. Plus précisément, Cooper (1989) a montré que dans le pire des cas l'algorithme permettant d'établir la k-cohérence, forte ou non, est en complexité temporelle $\mathcal{O}(n^k \times d^k)$. Cette complexité rend inutilisable l'application de la k-cohérence à chaque nœud de l'arbre de recherche dès lors que k est grand. Cependant, il est possible d'utiliser ces formes de cohérences en prétraitement au début de la recherche.

3.2.3.2 Les contraintes globales

Les contraintes globales, introduite par (Lauriere 1978), sont des contraintes classiques auquel les sont appliquées des algorithmes de filtrages spécifiques. Elles permettent d'englober plusieurs contraintes et de décrire de manière concise une relation. La première, et la plus célèbre est la contrainte d'exclusion mutuelle pour un ensemble de variables $\mathcal{X}$ ($AllDiff(\mathcal{X})$). Cette contrainte permet de préciser que les valeurs choisies pour les variables impliquées doivent toutes être différentes (Régin 1999). Ces contraintes permettent de conserver le sens de la contrainte, ce qui permet d'adapter l'algorithme de filtrage à chaque contrainte globale. En contrepartie de ce gain d'expressivité, les contraintes globales font appel à des techniques de filtrage beaucoup plus complexes que celles utilisées généralement. Néanmoins, les avantages apportés par une modélisation à l'aide de telles contraintes ne se limitent pas à des procédures de filtrage spécifiques. En effet, il existe des procédures de vérification capable de prouver l'inconsistance d'un problème. Il existe un catalogue de plus de 350 contraintes globales (Beldiceanu *et al.* 2005).

3.2.3.3 Choix heuristiques

Comme pour SAT, l'ordre avec lequel sont affectées les variables et la valeur selon laquelle elles doivent être assignées est très important et joue un rôle prépondérant en ce qui concerne la taille de l'arbre recherche (Bacchus et Run 1995, Gent *et al.* 1996, Ginsberg *et al.* 1990, Haralick et Elliott

1980). Malheureusement, décider du meilleur couple variable/valeur à assigner est aussi difficile que de répondre à la satisfaisabilité du CSP (Liberatore 2000). Comme pour SAT, il est nécessaire de recourir à des heuristiques afin d'estimer au mieux quelle est la prochaine branche à explorer.

Heuristique de choix de variables L'ordre selon lequel les variables sont assignées peut être obtenu statiquement, c'est-à-dire que l'ordre d'affectation est calculé *a priori* avant le commencement de la recherche. Cependant, puisqu'il est difficile de prévoir en amont l'aspect de l'arbre de recherche, l'application d'une telle approche est en pratique inefficace. Une autre approche consiste à exploiter différentes informations sur l'état courant ou passé du problème afin de déterminer la variable la plus prometteuse. Ces heuristiques sont dynamiques et s'appuient pour la plupart sur l'un des deux principes suivants : « *fail first* » (Haralick et Elliott 1980) et « *promise* » (Beck *et al.* 2004). Ces deux politiques orientent la recherche de façon complètement opposée. Le principe « *fail first* » tente de guider la recherche le plus vite possible vers un échec tandis que « *promise* » tente d'assigner le maximum de variables. Nous dressons une liste non exhaustive d'heuristiques de choix de variables dynamiques :

- **dom :** cette heuristique, proposée par Haralick et Elliott (1980), est la première heuristique dynamique. Elle utilise la taille du domaine des variables du réseau courant afin de discriminer la prochaine variable. Plus précisément, elle choisit la variable dont le domaine est le plus petit comme prochaine variable à assigner ;
- **ddeg :** proposée par Dechter et Meiri (1989), elle consiste à ordonner les variables de manière décroissante en fonction de leur degré courant. En d'autres termes, elle sélectionne en priorité la variable connectée avec le plus grand nombre de variables non encore assignées ;
- **wdeg :** l'heuristique *wdeg* (*Weighted degree*), proposée par Boussemart *et al.* (2004), consiste à choisir les variables apparaissant dans les contraintes les plus souvent falsifiées. Cette approche, expérimentée dans le cadre de SAT par Brisoux *et al.* (1999), consiste à associer un compteur à chaque contrainte. Ce compteur, initialisé à 1, est incrémenté lorsque la contrainte qui lui est associée est violée au cours de la recherche. Grâce à cette pondération, il est possible d'estimer les parties difficiles ou les noyaux inconsistants d'une instance CSP. Le degré pondéré d'une variable X est alors obtenu en sommant le poids des contraintes impliquant X et au moins une autre variable non assignée ;
- **combinaison heuristiques :** il est possible de combiner les trois heuristiques précédentes afin d'en obtenir de nouvelles. En pratique, combiner ces heuristiques permet d'obtenir de meilleures performances que de les exécuter séparément. Les heuristiques dom/deg (Bessière et Régin 1996) et $dom/ddeg$ (Smith et Grant 1997) sont, par exemple, en moyenne plus efficace que les heuristiques dom ou $ddeg$ seules. Elles consistent respectivement à choisir en priorité la variable avec le plus petit ratio taille du domaine courant sur le degré pris sur le réseau initiale et le ratio taille du domaine courant sur degré courant. À l'heure actuelle la combinaison d'heuristiques la plus robuste est $dom/wdeg$ (Boussemart *et al.* 2004, Lecoutre *et al.* 2004, Hulubei et O'Sullivan 2005). Elle consiste à sélectionner en priorité la variable avec le plus petit ratio taille du domaine courant sur degré pondéré courant.

Heuristique de choix de valeur Lorsqu'un schéma de branchements binaires est utilisé, il est nécessaire de sélectionner parmi l'ensemble des valeurs du domaine de la variable choisie par l'heuristique de choix de variables celle qui doit être assignée ou réfutée. Contrairement à SAT, le nombre de valeurs candidates peut être très important et sélectionner efficacement la plus appropriée est une tâche ardue. Il n'est pas rare qu'en pratique le choix de la prochaine valeur soit effectué de manière aléatoire ou en fonction de l'entier associé à la valeur (la plus petite valeur par exemple). Cependant, il est possible de trouver dans la littérature un certain nombre d'heuristiques. Nous en donnons ici une liste non exhaustive :

- Dechter et Pearl (1987) proposent une heuristique de choix de valeur statique basée sur une approximation du nombre de solutions de chaque sous-problème. Cette approximation est obtenue en relaxant le graphe de contraintes initial par la suppression de certaines contraintes jusqu'à obtenir un arbre. Une fois ce dernier obtenu, il est possible de comptabiliser le nombre de solutions d'un arbre en temps polynomial. Les valeurs sont alors ordonnées de manière décroissante en fonction du nombre de solution où elles apparaissent. D'autres travaux, s'appuyant sur cette approche, tentent d'estimer de manière dynamique ce nombre grâce à l'utilisation de réseaux Bayesiens (Kask *et al.* 2004, Meisels *et al.* 2000) ;
- Ginsberg *et al.* (1990) proposent d'estimer l'impact de l'assignation d'une valeur v de X sur le domaine des autres variables. Pour cela, ils calculent pour chaque variable Y du réseau la taille de $dom(Y)$ obtenue sur le réseau simplifié par la propagation de $(X = v)$. Plus précisément, étant donné un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$, ils définissent une fonction f telle que $f(Y, \mathcal{P})$ est le nombre de valeurs qui reste dans le domaine de Y après la propagation de contraintes effectuée sur $\mathcal{P}$. À partir de ce principe deux heuristiques vont voir le jour afin de sélectionner la valeur à assigner pour une variable X choisie. La première, appelée « *promise* » et proposée par Frost et Dechter (1995), consiste à choisir la valeur v qui maximise $\sum_{Y \in \mathcal{X}} f(Y, \mathcal{P}_{|(X=v)})$. La seconde, appelée « *min-conflict* » et proposée par Geelen (1992), maximise quant à elle le produit, c'est à dire $\prod_{Y \in \mathcal{X}} f(Y, \mathcal{P}_{|(X=v)})$;
- une des heuristiques les plus utilisées à ce jour est basée sur le même principe que l'heuristique de choix de variables $wdeg$. Elle s'appuie sur une analyse du passé afin de déterminer quelle est la valeur qui a la plus forte probabilité de conduire à un noyau minimalement incohérent. Pour cela, un compteur est associé à chaque valeur du problème. Lorsqu'un conflit est atteint, les valeurs ayant conduit à cet échec sont incrémentées. La valeur choisie est la valeur qui maximise ce poids ;
- il est aussi possible d'étendre certaines heuristiques utilisées dans le cadre de SAT. Par exemple, Lecoutre *et al.* (2007) proposent d'étendre l'heuristique de choix de valeurs JW (Jeroslow et Wang 1990) au problème CSP.

3.2.3.4 Randomisation et Redémarrages

Comme nous l'avons déjà énoncé dans le cadre de SAT (voir section 3.1.5.3), les algorithmes de *backtracking* sont très sensibles aux choix effectués au sommet de l'arbre. En effet, il peut y avoir de forte variabilité de performances en fonction de l'ordre selon lequel les variables sont assignées. Afin de tirer partie de ce phénomène deux techniques ont été proposées. Ces techniques, appelées *randomisation* et *redémarrages*, permettent de parcourir l'espace de recherche de manière différente.

Plusieurs méthodes ont été proposées afin d'ajouter du « bruit » au sein de l'algorithme de *backtracking*. Harvey (1995) propose, par exemple, d'effectuer des permutations aléatoires dans l'ordre de choix de variables ou de valeurs. Gomes *et al.* (1998) proposent de randomiser l'ordre d'assignation des variables en modifiant la stratégie permettant de départager les ex aequo (par exemple, pour l'heuristique dom deux variables ayant la même taille de domaine sont une fois départagées avec l'ordre lexico et une autre fois avec l'ordre lexico inverse) ou en choisissant la variable qui a le moins de chance d'être sélectionnée (par exemple, la variable avec le plus grand domaine pour l'heuristique dom). D'autres alternatives consistent à choisir la variable avec une certaine probabilité donnée par le poids de la variable ou de choisir aléatoirement, parmi un portfolio d'heuristiques, une nouvelle heuristique pour chaque point de choix.

Afin d'être efficace il est nécessaire que ces approches soient effectuées le plus haut possible dans l'arbre de recherche. Pour effectuer cela une méthode simple consiste à effectuer périodiquement un redémarrage. Comme pour SAT, cette approche consiste à arrêter l'évaluation de l'interprétation courante et

à reprendre le problème depuis la racine de l'arbre de recherche tout en conservant certaines informations, telles que le poids des contraintes fournies par l'heuristique $wdeg$. Ces politiques de redémarrages sont calculées en fonction du nombre de conflits rencontrés et sont semblables aux heuristiques statiques présentées dans le cadre de SAT (voir 3.1.5.3).

3.2.4 Synthèse

Dans cette section, nous avons vu qu'il était possible d'utiliser les propriétés des contraintes pour réduire l'espace de recherche. Ces méthodes, appelées cohérence locale, combinées avec un simple algorithme de *backtracking* permettent d'obtenir des algorithmes de résolution complets efficaces. Néanmoins, il est très rare que des formes de cohérences plus fortes que la cohérence d'arc soit maintenue à chaque nœud de l'arbre de recherche. En effet, elles sont pour la plupart difficiles à maintenir et nécessitent souvent la mise en place de structures de données complexes et gourmandes en mémoire. Par conséquent, ces méthodes sont, au mieux, utilisées pour le prétraitement de la formule.

3.3 Conclusion

Nous venons de présenter différents paradigmes pour résoudre de manière exacte les problèmes SAT et CSP. Comme nous l'avons déjà fait remarqué initialement, ces méthodes s'appuient sur une recherche arborescente et utilisent des méthodes de propagation de contraintes afin de couper les branches ne pouvant conduire à une solution. Cependant, les similitudes ne s'arrêtent pas là puisque quelque soit le formalisme utilisé ces méthodes se comportent de la même manière pour tous types de problèmes. En effet, que ce soit en SAT ou en CSP, ces approches sont très efficaces sur les instances structurées et très mauvaises sur les instances aléatoires. De plus, du fait de leur nature exponentielle, il s'avère que sur certaines instances de grandes tailles elles soient inefficaces et ne permettent de fournir aucune information sur le problème posé.

Comme nous pouvons le remarquer, les avantages et les inconvénients des approches de *backtracking* sont respectivement les inconvénients et les avantages des méthodes de recherche locale. En effet, ces dernières sont efficaces sur les instances aléatoires et inefficaces sur les instances structurées, elles peuvent travailler sur les instances de grande taille mais sont incapables de prouver l'inconsistance d'une formule. En partant de ce constat, il parait intéressant d'hybrider ces deux types d'approches afin d'en combiner les avantages. L'élaboration de telles approches est discutée dans le chapitre suivant.

Chapitre 4

Recherche Locale et méthodes complètes : hybridations

Sommaire

COMME nous l'avons vu lors de la présentation des algorithmes incomplets (voir chapitre 2) et des algorithmes complets (voir chapitre 3), la recherche locale et les algorithmes de *backtracking* sont complémentaires. Partant de ce constat, il semble tout à fait naturel de s'interroger sur la faisabilité d'une approche permettant de tirer avantage de chacun de ces deux paradigmes de résolution. Une telle combinaison a même été élevée au rang de défi par la communauté SAT. En effet, Selman *et al.* (1997) proposent dix challenges dont le septième expose le problème suivant :

> « `Demonstrate the succesful combination of stochastic local search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.` »

Le but d'une méthode hybride est d'exploiter les qualités des différentes approches utilisées en son sein. Une hybridation parfaite serait une méthode meilleure que les deux approches prises séparément. Dans ce chapitre, nous présentons différents schémas d'hybridation et certains exemples de solveurs les utilisant.

4.1 Schémas de combinaisons existants

Depuis quelques années, beaucoup d'approches ont été proposées afin de faire coopérer méthodes complètes et incomplètes. Ces dernières peuvent être classifiées en fonction de la manière dont les deux méthodes interagissent. Nous distinguons généralement deux catégories d'approches hybrides, comme l'illustre la figure 4.1 :

– *combinaisons collaboratives* : par collaboration nous entendons que les algorithmes échangent des informations et qu'ils sont exécutés de manière indépendante séquentiellement ou en parallèle. Une manière d'implémenter ce schéma consiste par exemple à exécuter une des deux méthodes en prétraitement de l'autre ;

– *combinaisons intégratives* : par intégration nous entendons que l'une des méthodes est une subordonnée ou une composante d'une autre méthode. Ainsi, nous distinguons un « maître », pouvant être soit une méthode complète soit une méthode incomplète, possédant au moins un « esclave ». Par exemple, la recherche locale peut aider un algorithme de *backtracking* soit comme heuristique de branchement soit afin d'étendre l'interprétation courante en une solution. Dans d'autre cas, la recherche locale peut être vue comme le processus principal et la propagation de contraintes peut être utilisée pour limiter l'espace de recherche ou pour capturer la structure du problème. D'autres approches consistent à utiliser une méthode complète pour explorer le voisinage d'un minimum local afin de s'en extraire.

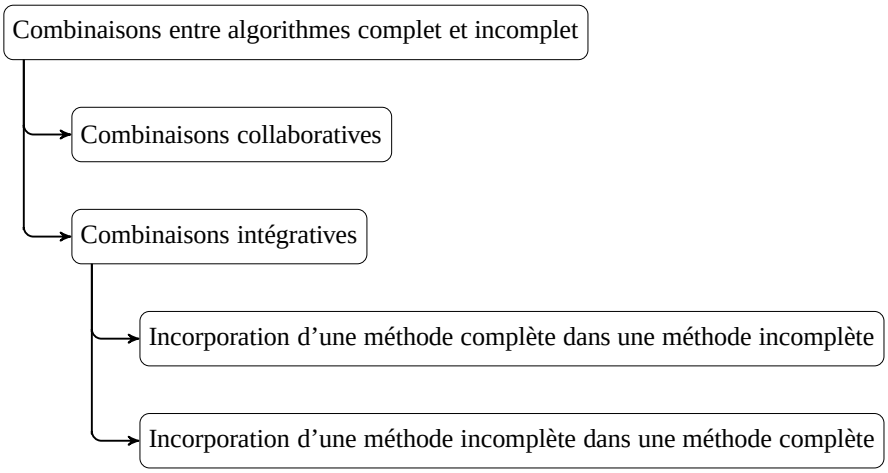


FIGURE 4.1 – Classification des méthodes hybrides.

Dans les deux sections suivantes, cette classification est détaillée et des exemples de solveurs issus de la littérature, portant aussi bien sur le domaine SAT que CSP, sont présentés.

4.2 Hybridations collaboratives

Dans cette section nous présentons trois approches où la combinaison entre la recherche locale et les méthodes complètes est de haut niveau (c'est-à-dire que aucun algorithme n'est inclus dans un autre). La première consiste à utiliser une des deux méthodes en prétraitement de l'autre (Crawford 1993, Eisenberg et Faltings 2003, Pham *et al.* 2007b, Paris *et al.* 2007, Mouhoub et Jafari 2011), la seconde exécute de manière alternée les deux approches (Vion 2007) et la dernière consiste à résoudre des sous-problèmes, obtenus à l'aide d'une recherche locale, grâce un algorithme complet (Zhang et Zhang 1996, Fang et Hsiao 2007, Letombe et Marques-Silva 2008).

4.2.1 Prétraiter à l'aide de l'une des deux méthodes

Une manière simple de faire collaborer deux approches est d'en exécuter une en prétraitement de l'autre. La première approche présentée consiste à utiliser la recherche locale en prétraitement d'une méthode complète. En effet, comme nous l'avons souligné et comme l'ont montré Grégoire *et al.* (2007), pour la détection de MUS, la recherche locale fournie une bonne heuristique en ce qui concerne la détection de noyaux insatisfiables. Il semble donc naturel d'utiliser cette dernière en prétraitement afin d'identifier les parties difficiles de l'instance à résoudre. C'est sur ce principe que sont basées les approches

proposées par Crawford (1993), dans le cadre de SAT, Eisenberg et Faltings (2003) et plus récemment Mouhoub et Jafari (2011), dans le cadre CSP. La recherche locale est exécutée sur le problème initial pendant un certain laps de temps afin de pondérer les clauses susceptibles d'appartenir à un noyau minimalement inconsistant. Une fois cet intervalle de temps écoulé, le poids obtenu au niveau des contraintes est utilisé afin d'ordonner les variables en vue d'un parcours systématique de l'espace de recherche. La figure 4.2 illustre un tel schéma d'hybridation.

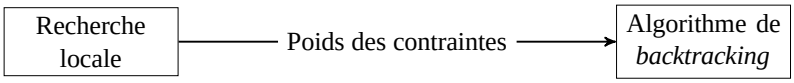


FIGURE 4.2 – La recherche locale comme prétraitement d'une méthode complète.

Une autre manière de prétraiter un problème consiste à utiliser une des deux méthodes afin d'extraire des informations structurelles ou de produire un sous-problème plus « facile » à résoudre. Pham *et al.* (2007b) proposent, dans le cadre de SAT, une approche utilisant une méthode complète (Ostrowski *et al.* 2002) afin d'extraire de la formule des dépendances fonctionnelles. Une fois ces dernières récoltées, un graphe de dépendances est généré et une méthode de recherche locale est exécutée sur le sous-ensemble des variables indépendantes. Cependant, afin d'exploiter au mieux la structure du problème la fonction de coût est redéfinie de manière à considérer les portes préalablement calculées. Pour cela, chaque contrainte conserve les variables qui, si elles sont inversées, changent la valeur de vérité associée à la porte. Pour définir le nombre de portes satisfaites (respectivement falsifiées) après l'inversion de la valeur de vérité d'une variable il suffit alors de comptabiliser le nombre de contraintes falsifiées (respectivement satisfaites) où celle-ci apparaît.

Exemple 4.1. *Considérons le graphe de dépendances illustré dans la figure 4.3 obtenu à partir de la formule* $\Sigma = \{(\neg g_1 \vee v_2 \vee v_3), (g_1 \vee \neg v_2), (g_1 \vee \neg v_3), (g_2 \vee \neg v_2 \vee \neg v_3), (\neg g_2 \vee v_3), (\neg g_2 \vee v_4), (g_3 \vee g_1 \vee g_2), (\neg g_3 \vee \neg g_1 \vee g_2), (\neg g_3 \vee g_1 \vee \neg g_2), (g_3 \vee \neg g_1 \vee \neg g_2), (v_1 \vee g_1)\}$. *L'ensemble* $\{v_1, v_2, v_3, v_4\}$ *représente les variables indépendantes du problème. À partir du graphe il est possible de déterminer qu'inverser la valeur de vérité de* v_2 *permet de satisfaire la contrainte* α *ainsi que les deux portes* g_1 *et* g_3 *tandis que flipper* v_3 *satisfait la porte* g_2 *mais falsifie* g_3.

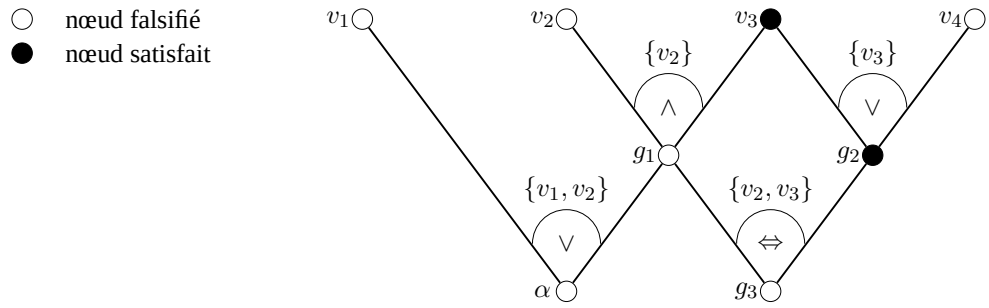


FIGURE 4.3 – Utilisation des dépendances fonctionnelles au sein d'une recherche locale.

Toujours dans le cadre de SAT, Paris *et al.* (2007) proposent d'utiliser la recherche locale afin de fournir un ensemble *strong backdoor* [19] de taille raisonnable pour guider une méthode complète. Pour ce faire, les auteurs proposent une utilisation originale d'une méthode de recherche locale afin de fournir

19. Soit Σ une CNF, un ensemble de variables $\mathcal{B}$ est un ensemble *backdoor* s'il existe une interprétation des variables de $\mathcal{B}$ telle que la formule Σ simplifiée appartient à une classe polynomiale de SAT. Cet ensemble est dit *strong backdoor* si pour toute interprétation des variables de $\mathcal{B}$ la formule simplifiée appartient à une classe polynômiale pour le problème SAT.

un renommage maximisant la sous-formule horn-renommable d'une CNF donnée. Plus précisément, ils proposent de modifier la fonction objective de telle sorte que la fonction à minimiser soit le nombre de clauses non-horn. La redéfinition de cette dernière permet alors d'utiliser les techniques de recherche locale traditionnelles (exemple : NOVELTY, WALKSAT, ...) afin d'obtenir une méthode permettant de définir un renommage minimisant la partie non-horn de la formule. Ensuite, à partir de la sous-formule non-horn obtenue, un ensemble *strong backdoor* est extrait. Cet ensemble est alors utilisé au sein d'un solveur CDCL afin de guider l'heuristique de choix de variables.

4.2.2 Exécution alternée

Vion (2007) propose une approche hybride qui consiste à alterner l'exécution d'une recherche locale et d'un algorithme de type MAC. Dans cette dernière, illustrée dans la figure 4.4, les deux approches sont exécutées séquentiellement pendant un certain laps de temps tout en se transmettant des informations. Plus exactement, la recherche locale effectue un certain nombre de réparations pendant lesquels des informations sur le problème sont extraites (pondération des clauses falsifiées). Lorsque le nombre de *flips* maximal est atteint et qu'aucune solution a été trouvée, l'ensemble des informations collectées est transmis à un algorithme de *backtracking* qui prend alors le relais. Ce dernier est exécuté de la même manière tant qu'un certain nombre de conflits n'a pas été atteint, si une solution ou l'absence de solution est démontrée le résultat est retourné, sinon la recherche locale reprend la main et ainsi de suite. Un des avantages de cette approche est que, s'il existe une meilleure stratégie pour un problème donné (par exemple, la recherche locale pour les problèmes aléatoires difficiles, ou la recherche systématique pour les problèmes industriels), le temps perdu à exécuter l'autre approche est limité.

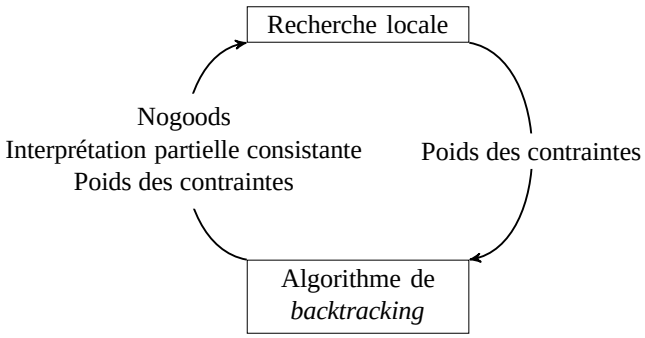


FIGURE 4.4 – Exécution alternée d'une recherche locale et d'un algorithme de *backtracking*.

4.2.3 La recherche locale pour extraire un sous-problème inconsistant

Nous présentons ici deux méthodes où la recherche locale est utilisée pour extraire un sous-problème qui est ensuite fourni à une méthode complète afin d'être résolu. La première est une approche proposée par Zhang et Zhang (1996) et est décrit dans l'algorithme 4.1. Elle consiste à exécuter une recherche systématique sur le sous-problème induit par une interprétation partielle. Plus précisément, étant donnée une formule Σ, la méthode commence par choisir aléatoirement une interprétation partielle $\mathcal{I}$ (ligne 2), puis une méthode complète de type DPLL tente de résoudre le sous-problème $\Sigma_{|\mathcal{I}}$ (ligne 4). Si cette dernière échoue, l'interprétation partielle est réparée à l'aide d'une stratégie de recherche locale (ligne 5) et le processus est itéré. Cette méthode, du fait qu'elle travaille sur un sous-problème, n'est pas capable de prouver l'inconsistance de la formule et est par conséquent incomplète.

Algorithme 4.1 : Approche hybride de Zhang et Zhang (1996)

Données : Σ un ensemble de contraintes et $nbRep$ le nombre de réparations autorisées par la RL
Résultat : `vrai` si le réseau est satisfiable

```
1 Début
2     I_p ⟵ interprétation partielle générée de manière aléatoire;
3     tant que vrai faire
4         si DPLL(Σ_{|I_p}) alors retourner vrai;          /* Σ est cohérent */
5         RL(Σ, I_p, nbRep)
6 Fin
```

La seconde approche, proposée par Fang et Hsiao (2007) et revisitée par Letombe et Marques-Silva (2008), est décrite dans l'algorithme 4.2. Elle consiste à utiliser la recherche locale afin de construire incrémentalement un ensemble de clauses qui est ensuite résolu par une approche de type DPLL. Plus précisément, soient Σ un problème à résoudre, Ψ un ensemble de contraintes initialisé à vide et $\mathcal{I}$ une interprétation utilisée par une méthode de recherche locale. La recherche locale commence par rechercher une solution au problème Σ (ligne 4). Après un temps donné, si aucune solution n'est trouvée, l'ensemble de clauses falsifiées par l'interprétation $\mathcal{I}$ est ajouté à Ψ (ligne 6). Un solveur de type DPLL est ensuite exécuté sur la sous-formule Ψ. Si ce sous-problème est montré incohérent, le problème initial est aussi montré incohérent (ligne 9). Sinon, $\mathcal{I}$ est réparée à l'aide du modèle trouvé (lignes 7-8) et la recherche locale est de nouveau exécutée. Ce processus est réitéré tant qu'une solution ou que l'absence de solution n'a pas été démontrée (lignes 2-9). Afin de rendre la méthode complète, une clause $\beta \in \Sigma \setminus \Psi$, choisie de manière aléatoire, est ajoutée à Ψ avant chaque nouvelle itération de l'approche complète (ligne 9).

Algorithme 4.2 : Approche hybride de Fang et Hsiao (2007)

Données : Σ un ensemble de contraintes et $nbRep$ le nombre de réparations autorisées par la RL
Résultat : `vrai` si le réseau est satisfiable, `faux` sinon

```
1  Début
2      I ⟵ interprétation complète générée de manière aléatoire ; Ψ ⟵ ∅;
3      tant que vrai faire
4          RL(Σ, I, nbRep);
5          si I ⊨ Σ alors retourner vrai;                  /* Σ est cohérent */
6          Ψ ⟵ Ψ ∪ {α ∈ Σ telle que I ⊭ Σ} ∪ {β} telle que β ∈ Σ \ Ψ;
7          si DPLL(Ψ) alors
8              I ⟵ I' ∪ {ℓ ∈ I telle que ℓ ∉ I' et ℓ̃ ∉ I'} où I' est le modèle trouvée;
9          sinon retourner faux;                           /* Σ est incohérent */
10 Fin
```

4.3 Hybridations intégratives

Dans cette section nous exposons une liste non exhaustive de solveurs hybrides de type intégratifs. Pour présenter ces méthodes nous employons la notion de souche qui caractérise la base de la méthode à savoir, complète ou incomplète. Par exemple, si nous utilisons une recherche locale pour trouver la variable de branchement d'un algorithme systématique, nous considérons que la souche est de type

complète.

4.3.1 Complet greffé sur incomplet

Cette catégorie regroupe les schémas de coopération où l'utilisation de mécanismes empruntés aux méthodes complètes est utilisée afin de modifier ou d'orienter une méthode incomplète. Nous décrivons ici deux approches basées sur ce principe. La première consiste à ajouter des contraintes à la formule afin de la rendre complète (Ginsberg et McAllester 1994, Cha et Iwama 1996, Fang et Ruml 2004). La seconde est basée sur une utilisation des processus de propagation de contraintes (Balint *et al.* 2009, Gableske et Heule 2011, Vasquez et Dupont 2002, Jussien et Lhomme 2002, Havens et Dilkina 2004).

4.3.1.1 Ajouts de contraintes

Cette méthode a d'abord été initiée dans le cadre de CSP par Ginsberg et McAllester (1994) et ensuite par Cha et Iwama (1996) dans le cadre de SAT. Cette approche, décrite dans l'algorithme 4.3, est basée sur une recherche locale où la stratégie d'échappement consiste à ajouter des contraintes à la formule initiale lorsqu'un minimum local est atteint. L'ajout de telles contraintes permet à la fois de sortir du minimum local, par redéfinition de la fonction objectif, et de pouvoir montrer l'incohérence de la formule. Ce principe a ensuite été exploré et amélioré par Fang et Ruml (2004) et Shen et Zhang (2005) dans le cadre de SAT. Néanmoins, la principale différence entre toutes ces approches réside dans la manière dont sont générées les contraintes à ajouter. Dans la méthode proposée par Ginsberg et McAllester (1994) les contraintes ajoutées correspondent à des « nogood » générés à partir d'une contrainte falsifiée par l'interprétation courante à l'aide de la méthode introduite dans l'article de McAllester (1993). Dans le cadre de SAT, les contraintes ajoutées correspondent à une (Cha et Iwama 1996) ou plusieurs (Fang et Ruml 2004, Shen et Zhang 2005) clauses générées par résolution à partir de clauses falsifiées par l'interprétation courante lorsqu'un minimum local est atteint. Le point commun de ces trois approches est que les clauses générées sont toujours obtenues à partir d'une seule résolvante. Dans le chapitre 6 nous proposons d'étendre ce concept en considérant un chemin de résolutions. Plus précisément, nous proposons une approche basée sur la construction et l'analyse d'un graphe d'implications comme cela est fait dans les méthodes complètes de type CDCL (Zhang *et al.* 2001).

Algorithme 4.3 : Ajout de contraintes au sein d'une recherche locale.

Données : $\mathcal{F}$: un ensemble de contraintes
Résultat : `vrai` si le problème est satisfiable, `faux` sinon

```
1  Début
2      I ⟵ interprétation complète générée de manière aléatoire;
3      tant que vrai faire
4          tant que (∃I' ∈ N(I) permettant une descente) faire
5              I ⟵ I'
6          si (I ⊨ F) alors retourner vrai ;
7          Soit Ψ un ensemble de contraintes généré par résolution à partir de clauses falsifiées par I;
8          si ⊥ ∈ Ψ alors retourner faux ;
9          F ⟵ F ∪ Ψ;
10 Fin
```

4.3.1.2 Filtrer l'espace de recherche

Si la recherche locale utilise les contraintes pour définir une fonction d'évaluation, elle peut utiliser la formulation de ces contraintes et exploiter le fait qu'elles puissent réduire l'espace de recherche, notamment grâce aux algorithmes de propagation de contraintes telles que la propagation unitaire (voir section 3.1.4.1) dans le cadre de SAT, et l'arc-cohérence (voir section 3.2.3.1) dans le cadre CSP. Nous présentons ici deux approches s'appuyant sur ce principe.

La première approche est une méthode nommée UNITWALK et a été introduite par Hirsch et Kojevnikow (2005) dans le cadre de SAT. Le schéma général, décrit dans l'algorithme 4.4, ressemble à celui de WALKSAT, excepté que chaque essai commençant avec une affectation aléatoire différente est constitué de périodes. Pendant une période, au moins un flip (souvent beaucoup plus) est effectué. L'algorithme commence par choisir une permutation aléatoire des variables (ligne 4). Ensuite, il effectue une copie de la formule initiale (ligne 5) et simplifie, à chaque pas de recherche, la formule avec une méthode de propagation unitaire (lignes 6-13). La seule différence entre cette propagation unitaire et celle utilisée par les méthodes exactes réside dans le fait que l'interprétation étant déjà complète, la valeur des variables apparaissant dans les clauses unitaires n'est flippée que si la clause était fausse (ligne 9). Lorsque plus aucune clause unitaire n'apparaît, la formule est simplifiée en prenant la variable de la permutation suivante et sa valeur dans l'affectation courante comme paramètres (ligne 11). Si une période s'est déroulée sans aucun flip, une variable est choisie aléatoirement et sa valeur est inversée pour que la période suivante soit différente de la dernière effectuée (ligne 13-14).

Algorithme 4.4 : UNITWALK

Données : Σ une CNF telle $\mathcal{V}_\Sigma = \{x_1, x_2, \ldots x_n\}$et deux entiers $maxEssais$ et $maxPeriodes$
Résultat : `vrai` si le problème est satisfiable, `faux` s'il est impossible de conclure

```
1  Début
2      pour t de 0 à maxEssais faire
3          I ⟵ une interprétation complète générée aléatoirement;
4          π = permutation aléatoire de 1 ... n;
5          Ψ = Σ;
6          pour p de 0 à maxPeriodes faire
7              pour i de 0 à n faire
8                  pour chaque α ∈ Ψ telle que |α| = 1 faire
9                      I ⟵ (I \ ¬α) ∪ α;                  /* ajuster I en fonction de α */
10                     Ψ ⟵ Ψ|α;
11                 Ψ ⟵ Ψ|x_π[i];                          /* propager π[i] variable de V_Σ */
12             si (Ψ = {}) alors retourner vrai ;
13             si aucun flip n'a été effectué pendant la péritode alors
14                 flipper de manière aléatoire une variable dans I;
15 Fin
```

La seconde approche consiste à utiliser la propagation de contraintes afin de vérifier et ajuster l'interprétation complète utilisée par une recherche locale. Cette approche, décrite dans l'algorithme 4.5, ne diffère d'une recherche locale classique que sur la manière dont les minima locaux sont gérés. En effet, dans cette dernière le critère d'échappement classique n'est utilisé que dans certaines conditions. Plus précisément, deux fonctions `PROPAGATION` et `CONDITION`, permettant respectivement d'effectuer un traitement sur l'interprétation complète et de savoir quand effectuer ce traitement, sont introduites. Lorsqu'un

minimum local est atteint (ligne 7) un appel à la fonction `CONDITION` est effectué. Si ce dernier retourne `vrai` alors l'interprétation courante est modifiée à l'aide de la fonction `PROPAGATION` (ligne 8), sinon un critère d'échappement classique est utilisé (ligne 9). Cette méthode dépend donc essentiellement des fonctions `CONDITION` et `PROPAGATION`.

Algorithme 4.5 : Utilisation de la propagation de contraintes pour guider la recherche locale

Données : $\mathcal{F}$ une formule, deux entiers $maxReparations$ et $maxEssais$ ainsi que deux fonctions `PROPAGATION` et `CONDITION` permettant respectivement d'effectuer un traitement sur l'interprétation complète et de savoir quand effectuer ce traitement
Résultat : `vrai` si la formule $\mathcal{F}$ est satisfiable, `faux` s'il est impossible de conclure

```
1  Début
2    pour i de 0 à maxEssais faire
3      I ⟵ une interprétation complète générée aléatoirement;
4      pour j de 0 à maxReparations faire
5        si (eval(F, I) = 0) alors retourner vrai ;
6        si (∃I' ∈ N(I) telle que diff(F, I, I') > 0) alors I ⟵ I';
7        sinon
8          si CONDITION() = vrai alors I ⟵ PROPAGATION(F, I);
9          sinon I ⟵ interprétation choisie suivant un critère d'échappement;
10   si (eval(F, I) = 0) alors retourner vrai ;
11   retourner faux ;
12 Fin
```

Dans le cadre de SAT, Balint *et al.* (2009) proposent une approche basée sur ce principe, nommée HYBRIDGM, et qui combine le solveur de recherche locale GNOVELTY+ (Pham *et al.* 2007a) (vainqueur de la compétition SAT 2007 catégorie aléatoire satisfiable) et le solveur complet MARCH$_{ks}$ (Heule et van Maaren 2006) (vainqueur de la compétition SAT 2007 catégorie aléatoire insatisfiable). Cette approche, spécialisée pour les instances aléatoires, consiste à utiliser une méthode de recherche locale afin de résoudre le problème tout en construisant une interprétation partielle $\mathcal{I}_p$ « prometteuses ». Lorsque $|\mathcal{I}_p|$ atteint une certaine taille (`CONDITION` = vrai), MARCH$_{ks}$ est exécuté sur la formule initiale simplifiée par l'interprétation $\mathcal{I}_p$. Si MARCH$_{ks}$ trouve une solution celle-ci est retournée, sinon le sous-problème est montré incohérent et l'interprétation partielle est modifiée (rôle de `PROPAGATION`). Finalement, les paramètres de GNOVELTY+ sont mis-à-jour et le processus est itéré tant qu'une solution n'a pas été trouvée ou que le nombre de flips maximal n'a pas été atteint. Du fait que seuls des sous-problèmes sont explorés de manière complète, cette méthode est incomplète. Afin de la rendre complète Gableske et Rüth (2010) proposent une approche divisée en deux phases. Dans la première, HYBRIDGM est exécuté pendant un certain laps de temps pendant lequel un ensemble d'informations est collecté et une solution est recherchée. Une fois ce temps écoulé une méthode complète est exécutée avec les informations récoltées pendant la première phase.

Gableske et Heule (2011) proposent par la suite d'utiliser ce schéma avec l'algorithme de recherche locale G^{2}WSAT. Lorsqu'un minimum local est atteint et qu'un certain nombre de flips a été effectué (déterminé à l'aide d'une distribution probabiliste de Cauchy) la fonction `PROPAGATION` est exécutée. Cette dernière consiste alors à affecter et à propager les littéraux de $\mathcal{I}$ tant que la clause vide n'a pas été générée ou que la formule n'est pas satisfaite (tous les littéraux sont propagés). L'interprétation partielle $\mathcal{I}'$ ainsi obtenue est alors utilisée afin d'ajuster l'interprétation $\mathcal{I}$ ($\mathcal{I} \longleftarrow (\mathcal{I} \setminus \mathcal{I}') \cup \mathcal{I}'$).

Dans le cadre de CSP, Vasquez et Dupont (2002) proposent de coupler ce schéma d'hybridation avec

la méthode TABOU. La fonction `CONDITION` implantée retourne toujours vrai. La liste tabou est quant à elle gérée grâce à la fonction `PROPAGATION`. Lorsqu'un minimum local est atteint une nouvelle variable est sélectionnée afin d'être propagée. Si après cette propagation le problème est incohérent, la méthode retourne la meilleure solution trouvée, sinon la recherche locale est de nouveau exécutée sur les variables non fixées par l'interprétation partielle.

Toujours dans le cadre de CSP, Jussien et Lhomme (2002) proposent une approche hybride, basée sur un schéma très similaire, où une recherche TABOU est utilisée afin d'explorer une interprétation partielle qui, pour une certaine forme de filtrage, n'admet pas de solution. Plus précisément, étant donnée une formule $\mathcal{F}$, une interprétation partielle $\mathcal{I}$ (aussi appelé *path*) est construite de manière incrémentale. Pour ce faire, tant qu'une solution n'a pas été trouvée, une nouvelle variable est assignée ($\mathcal{I} \longleftarrow \mathcal{I} \cup (X = v)$) et une méthode de filtrage ϕ est appliquée $\phi(\mathcal{F}_{|\mathcal{I}})$. Trois cas peuvent alors survenir : (i) $\phi(\mathcal{F}_{|\mathcal{I}})$ retourne un réseau où chaque variable est singleton et donc $\mathcal{F}$ est satisfiable ; (ii) $\phi(\mathcal{F}_{|\mathcal{I}})$ est consistant, dans ce cas $\mathcal{I}$ est étendue et le processus est itéré ; sinon (iii) $\phi(\mathcal{F}_{|\mathcal{I}})$ est inconsistant et une recherche locale est utilisée sur l'interprétation $\mathcal{I}$ pour la réparer, une fois réparée la méthode tente d'étendre à nouveau $\mathcal{I}$. Havens et Dilkina (2004) proposent un schéma d'hybridation similaire qui diffère principalement sur les méthodes de recherche locale utilisées.

4.3.2 Incomplet greffé sur complet

L'efficacité d'une méthode complète réside aussi bien dans le choix de la variable à affecter que dans les outils dont elle se dote pour élaguer l'arbre de recherche. Nous présentons ici deux approches permettant respectivement de choisir la prochaine variable à affecter (Mazure *et al.* 1998, Fourdrinoy *et al.* 2005, Ferris et Froehlich 2004, Nareyek *et al.* 2004) et de limiter l'espace de recherche exploré (Habet *et al.* 2002).

4.3.2.1 Recherche locale utilisée comme heuristique de branchement

Comme nous l'avons déjà souligné à plusieurs reprises, la recherche locale permet de mettre en exergue les parties difficiles d'un problème. Partant de ce constat, il semble naturel d'utiliser une telle approche afin de guider une méthode exacte de type *backtracking*. En effet, afin d'élaguer au maximum l'arbre de recherche, il est important d'affecter en priorité les variables jouant un rôle important dans l'insatisfiabilité de la formule étudiée.

Ce type d'approche a pour la première fois été exploré par Mazure *et al.* (1998) dans le cadre de SAT. Les auteurs proposent d'utiliser à chaque point de choix une heuristique de type recherche locale permettant soit d'étendre l'interprétation courante en une solution du problème, soit d'identifier la variable la plus souvent falsifiée. Afin d'obtenir une telle variable, les auteurs associent un poids à chaque contrainte et exécutent une recherche locale de type TABOU pendant un certain laps de temps pendant lequel à chaque flip le poids des contraintes falsifiées est incrémenté. Une fois ce temps imparti écoulé la variable apparaissant dans les contraintes les plus falsifiées est sélectionnée. Cette hybridation, sans pour autant surpasser les performances des meilleurs DPLL et des meilleures recherches locales de l'époque, s'est avérée très performante. En effet, elle a permis de résoudre des instances qui posaient problèmes aussi bien à DPLL qu'à la recherche locale. Néanmoins, appliquer la recherche locale trop fréquemment ou pendant trop longtemps peut avoir des conséquences néfastes en ce qui concerne les performances d'une telle approche. Afin de pallier à ce problème Fourdrinoy *et al.* (2005) proposent d'appeler la recherche locale uniquement lorsque cela semble pertinent. Plus précisément, ils proposent d'utiliser la recherche locale jusqu'à une profondeur prédéfinie puis d'utiliser une heuristique de branchement classique au

delà. Pour ce faire un seuil est fixé par l'utilisateur, lorsque la profondeur de l'arbre est supérieure à ce dernier, la recherche locale n'est pas lancée et la prochaine variable est choisie à l'aide des derniers résultats obtenus.

Un autre point important concerne les informations pouvant être récoltées par la recherche locale. Dans l'approche proposée par Mazure *et al.* (1998) le poids des clauses est ajusté à chaque flips. Néanmoins, comme l'ont montré Fourdrinoy *et al.* (2005) cette stratégie est coûteuse et n'est pas forcément utile. En effet, puisque la recherche locale se caractérise par une phase de descente, les informations relatives à celle-ci ne sont *a priori* pas significatives. Afin de ne pas perdre de temps dans ces dernières, les auteurs proposent de ne pondérer les clauses falsifiées que lorsque l'interprétation courante utilisée par la recherche locale ne falsifie pas plus d'un certain nombre de clauses. Néanmoins, définir un tel seuil est difficile et dépend fortement du problème traité. Afin de palier ce problème, ils proposent une seconde approche qui consiste à ne pondérer les clauses que lorsqu'un minimum local est atteint.

Toujours dans le cadre de SAT, Ferris et Froehlich (2004) proposent une approche basée sur ce principe mais qui tente de focaliser la recherche sur la partie satisfiable du problème. Dans cette dernière la recherche locale est exécutée avec une certaine probabilité et les informations qu'elle collecte sont utilisées afin d'identifier l'ensemble des littéraux appartenant au *backbone* de la formule. Pour ce faire, ils proposent d'étudier la déviation de chacun des littéraux, c'est-à-dire le ratio entre le nombre de fois où un littéral est affecté positivement et le nombre de fois où il est affecté négativement lors des différentes exécutions de la recherche locale. L'heuristique de branchement consiste alors à choisir ces littéraux en priorité.

Dans le cadre de CSP, Nareyek *et al.* (2004) proposent une approche pour résoudre le problème bien spécifique d'ordonnancement d'ateliers (*Job Shop*). Contrairement aux approches classiques le schéma de branchement utilisé par la méthode complète consiste à ajouter une contrainte unaire afin de réduire le domaine des variables (exemple, $dom(X) = \{0 \dots 100\}$ un point de choix peut consister à considérer $X < 50$). La recherche locale est alors utilisée afin d'affiner les intervalles sélectionnés par le solveur complet. Pour cela, elle est exécutée pendant un certain laps de temps (1000 réparations sont autorisées) pendant lequel elle identifie les valeurs apparaissant le plus souvent dans les conflits. Une fois le temps imparti écoulé, la recherche locale propose une valeur parmi le sous-ensemble du domaine choisie comme point de choix.

4.3.2.2 Affaiblissement des méthodes complètes

La notion d'affaiblissement intervient dès lors qu'une méthode perd sa complétude. Les raisons, qui contraignent l'exploration à élaguer certaines branches de l'arbre de recherche, peuvent être liées à un temps d'exécution limité ou à la décision de concentrer la recherche sur une zone jugée plus intéressante et prometteuse.

Une manière de rendre un algorithme de *backtracking* incomplet est de limiter la profondeur de l'arbre de recherche pouvant être explorer. Ce principe de complétude partielle selon la profondeur à été introduit dans le cadre de SAT par Génisson et Rauzy (1996). Cependant, l'algorithme qu'ils proposent ne fait en aucune façon appel à un algorithme de recherche locale. Habet *et al.* (2002) proposent une méthode basée sur ce principe, nommée WALKSATZ et décrit dans l'algorithme 4.6, qui consiste à exécuter une méthode de recherche locale (WALKSAT) à chaque nouveau nœud de l'arbre de recherche construit par un algorithme de type DPLL (SATZ (Li et Anbulagan 1997)). Plus précisément, à chaque nœud de l'arbre de recherche construit par SATZ, jusqu'à une profondeur fixée (ligne 4), un graphe d'implications entre les littéraux de la formule est construit (ligne 5). Les composantes fortement connexes correspondant aux classes d'équivalences sont alors calculées et utilisées pour simplifier le graphe en

substituant les littéraux par leur unique représentant de la classe (ligne 7). Puis, la recherche de la fermeture transitive des implications est réalisée. Une fois ces étapes de simplification effectuées, les relations structurelles ainsi obtenues sont exploitées par le solveur WALKSAT (ligne 8).

Algorithme 4.6 : WALKSATZ

Données : Σ un ensemble de clauses, p la profondeur restant à explorer
Résultat : `vrai` si la formule est consistante, `faux` sinon

1 **Début**
2 $\Sigma \longleftarrow$ `SIMPLIFICATION(`Σ`);`
3 **si** $(\Sigma = \emptyset)$ **alors retourner** *vrai*;
4 **si** $(\bot \in \Sigma)$ *ou* $(p = 0)$ **alors retourner** *faux*;
5 Construire sur Σ le graphe des implications $\mathfrak{G}$ et son transposé $\mathfrak{G}^t$;
6 Propager l'état des variables sur $\mathfrak{G}$ et $\mathfrak{G}^t$;
7 Construire les classes d'équivalence et réduire Σ, $\mathfrak{G}$ et $\mathfrak{G}^t$;
8 **si** (`RL(`Σ, $\mathfrak{G}$, $\mathfrak{G}^t$`)`) **alors retourner** `vrai` ;
9 $\ell \longleftarrow$ `HeuristiqueDeBranchement` (Σ);
10 **retourner** (`WALKSATZ(`$\Sigma \wedge \ell, p-1$`)` *ou* `WALKSATZ(`$\Sigma \wedge \neg\ell, p-1$`)`)
11 **Fin**

4.4 Conclusion

Dans ce chapitre, différents schémas d'hybridation combinant une méthode de recherche locale et un algorithme de type *backtracking* ont été présentés. L'objectif affiché d'une telle démarche réside dans la volonté de combiner ces deux paradigmes de recherche afin de tirer parti des avantages de chacune tout en atténuant au mieux les inconvénients. Nous sommes en droit de nous poser la question de savoir si les approches présentées atteignent cet objectif et plus particulièrement si elles permettent de relever le challenge 7 proposé par Selman *et al.* (1997). Certains éléments de réponse à ces interrogations sont apportés dans (Kautz et Selman 2003). Dans cette article, les auteurs dressent un bilan des dix challenges proposés en 1997. En ce qui concerne le challenge qui nous intéresse, ils concluent que l'objectif n'est pas atteint. Bien que d'énormes progrès ont été réalisés, notamment sur la catégorie des instances aléatoires, nous pouvons sans trop nous avancer, convenir qu'à l'heure actuelle le défi n'est toujours pas relevé. En effet, les méthodes jusqu'alors développées n'atteignent pas les performances des meilleurs méthodes complètes sur les problèmes structurés. Néanmoins, comme nous allons le voir dans la partie II, cette piste de recherche n'est pas tarie.

Chapitre 5

Résolution parallèle du problème SAT

Sommaire

De par ses applications industrielles (planification, vérification formelle, bio-informatique, etc.) la résolution pratique du problème SAT est devenue très populaire. Cet engouement est dû, en majeure partie au fait que les problèmes pouvant être traités sont de plus en plus conséquents (des millions de variables et de clauses). Cette évolution s'explique à la fois par l'avènement des solveurs SAT modernes (*watched literal*, VSIDS, apprentissage) et par l'augmentation de la puissance de calcul.

Jusqu'en 2005, une méthode très simple afin de doubler les performances d'un solveur était d'attendre deux ans et de l'exécuter de nouveau sur un processeur de la nouvelle génération. En effet jusqu'à cette date, la puissance des processeurs augmentait en fonction du temps suivant approximativement une loi exponentielle. La puissance des nouvelles générations de processeurs doublait ainsi typiquement tous les deux à trois ans. Cette loi connue sous le nom de Loi de Moore (1998) se vérifie ainsi depuis les débuts de l'informatique, c'est-à-dire depuis environ quarante ans. Mais depuis 2005, cette loi souffre d'un petit ralentissement dû à des dissipations thermiques empêchant une augmentation de la fréquence des processeurs. Pour contourner ce problème, et ainsi garantir une augmentation de la puissance de calcul, les constructeurs ont modifié leur stratégie de fabrication. En effet, à l'heure actuelle, l'augmentation de la puissance de calcul se traduit par une augmentation du nombre d'unités de calcul. Ces nouvelles architectures impliquent implicitement une révolution dans la manière de concevoir les programmes.

Afin de pleinement profiter des nouvelles avancées technologiques, il est nécessaire que les solveurs SAT évoluent. Actuellement, deux types d'approches pour résoudre SAT en parallèle sont principalement utilisées. D'une part, les approches de type « diviser pour régner » consistent à partager l'arbre de recherche (« *guiding path* ») (Zhang *et al.* 1996, Sinz *et al.* 2001, Blochinger *et al.* 2003, Chrabakh et Wolski 2003, Chu et Stuckey 2008). D'autre part, les approches de type *portfolio* (Hamadi *et al.* 2009b, Guo *et al.* 2010, Biere 2010) mettent les solveurs en concurrence, permettant ainsi de résoudre une formule à l'aide de différentes stratégies. Dans ce chapitre, nous introduisons rapidement le concept d'architecture multi-cœurs (sans entrer dans les détails techniques) et les deux types d'approches utilisées afin de résoudre SAT en parallèle.

5.1 Résoudre SAT en parallèle

Nous introduisons à présent les concepts de base nécessaires à la compréhension des différents solveurs parallèles. En ce qui concerne les aspects technologiques liés au parallélisme, nous choisissons de ne pas les présenter. Néanmoins, le lecteur intéressé par ces aspects peut se référer à la thèse de Vander-Swalmen (2009) afin d'obtenir de plus amples informations.

5.1.1 Architectures multi-cœurs

Une architecture multi-cœurs peut-être vue comme un ensemble de processeurs ayant la possibilité de communiquer ensemble. Chaque processeur peut être vu comme une unité de calcul permettant d'exécuter une seule tâche à la fois.

Définition 5.1 (unité de calculs)**.** *Une unité de calculs $\mathcal{C}$ permet d'exécuter une tâche de manière séquentielle.*

Définition 5.2 (parallélisme)**.** *Le parallélisme en Informatique donne la possibilité d'exécuter plusieurs instructions simultanément.*

Exemple 5.1. *Considérons le problème lié au tri d'un tableau d'entiers. La figure 5.1 représente une exécution parallèle avec 4 unités de calcul.*

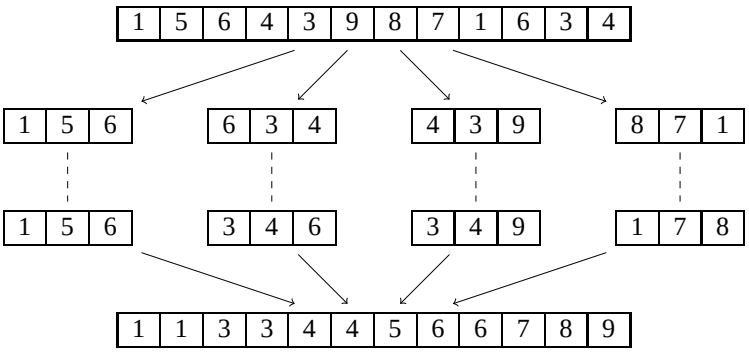


FIGURE 5.1 – Tri d'un tableau d'entiers à l'aide d'un algorithme parallèle.

Contrairement à l'exemple présenté précédemment, le problème SAT n'est pas facilement parallélisable. En effet, une caractéristique importante de ce problème est qu'il est très dur de prédire à l'avance le temps nécessaire à l'exploration d'une branche de l'arbre de recherche. Par conséquent, il est difficile (voire impossible) de partitionner de manière uniforme l'espace de recherche avant l'exécution du programme. Pour contourner ce problème il est nécessaire de dégager du parallélisme.

5.1.2 Dégager du parallélisme

Il existe principalement deux modèles permettant de faire travailler des processus en parallèle. Le premier, appelé *modèle collaboratif*, consiste à partager le travail entre l'ensemble des processus. Basé sur le principe « diviser pour régner », il constitue une méthode naturelle pour la conception et l'implémentation d'algorithmes pour de nombreux problèmes, et est particulièrement bien adapté aux approches récursives. Le second, appelé *modèle concurrentiel*, consiste à mettre en compétition plusieurs approches

sur le même problème. Ce dernier est très facile à mettre en place et est particulièrement bien adapté aux approches paramétrables. Dans la suite, nous présentons sommairement l'application de ces deux modèles pour la résolution parallèle de SAT.

5.1.2.1 Modèles collaboratifs

Les méthodes collaboritives sont pour la plupart basées sur le paradigme « diviser pour régner », qui consiste à réduire récursivement un problème en plusieurs sous-problèmes du même type. Du fait de leur conception, les algorithmes récursifs utilisent naturellement cette technique puisqu'ils s'appellent eux-mêmes une ou plusieurs fois sur une partition du problème initial et combinent les solutions pour retrouver une solution au problème initial.

Partant de ce principe, il semble naturel d'appliquer ce modèle de répartition de tâches dans le cas de solveurs DPLL. En effet, lorsque nous considérons l'arbre de recherche généré par une telle approche, celui-ci peut facilement être décomposé en plusieurs sous-arbres. Ce principe, appelé parallélisme par décomposition de domaines (figure 5.2), consiste à distribuer aux différents processus des sous-problèmes à calculer.

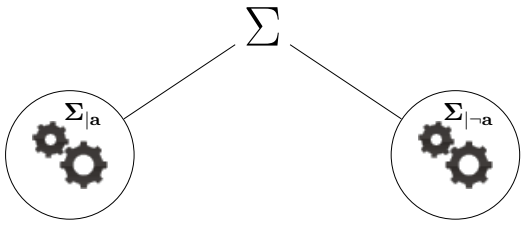


FIGURE 5.2 – Parallélisme par décomposition de domaines.

Cette méthode, bien que naturelle dans le cadre d'une approche DPLL, n'est pas facilement applicable pour les solveurs SAT modernes. En effet, comme le montre la figure 5.3, lorsqu'un problème est décomposé en deux sous-problèmes, il est possible que l'une des unités de calcul rencontre un conflit lui permettant d'effectuer un retour arrière non chronologique rendant le travail effectué par le deuxième processus inutile.

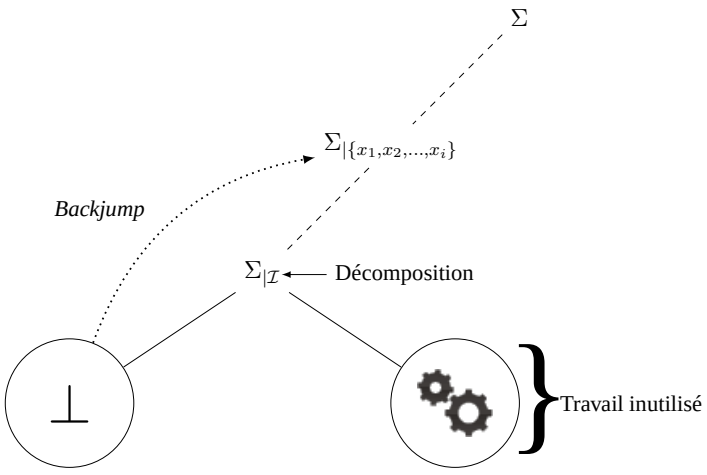


FIGURE 5.3 – Retour arrière non chronologique et travail inutile.

Un autre inconvénient de ce modèle réside dans le partage de la charge de travail entre les processeurs. En effet, puisque la taille des sous-arbres n'est pas connue à l'avance, il est nécessaire de mettre en place une politique d'équilibrage de charge entre les différentes unités de calcul de telle sorte qu'aucune ne reste oisive. Cet équilibrage peut être fait de manière statique ou dynamique.

L'équilibrage statique est une technique assez naturelle puisqu'elle consiste à distribuer les tâches avant que le processus de résolution ne débute. Cette approche, expérimentée par Böhm et Speckenmeyer (1996) dans le cadre de SAT, est difficilement applicable du fait de la difficulté à déterminer à *priori* la physionomie de l'arbre de recherche.

À l'opposé, l'équilibrage de charge dynamique (Livny et Melman 1982) consiste à distribuer les tâches aux processus en cours de recherche. Les charges de travail peuvent soit être mises en commun entre les processeurs périodiquement, soit être obtenues sur demande à l'aide de collecteurs de tâches. Ces collecteurs sont différents selon que l'équilibrage de charge est géré de manière centralisée ou distribuée.

Modèle centralisé Il comprend deux types de processeurs : le maître et les esclaves. Un processeur maître est une unité de calculs chargée de distribuer des tâches aux esclaves, qui sont chargés de les calculer. Dans ce modèle centralisé, le maître est responsable de l'équilibrage de charge des esclaves. Pour cela deux manières de procéder sont à distinguer :

- **liste centralisée** : le maître tient à jour une liste de tâches à effectuer à disposition des esclaves. Lorsqu'un esclave est inoccupé, il demande au maître du travail ;
- **liste distribuée** : chaque esclave tient localement une liste de tâches à traiter. Le maître a alors la charge de mettre en relation un esclave oisif et un esclave surchargé afin de transférer une partie de la charge de travail de l'esclave surchargé à l'esclave inoccupé.

Modèle distribué Comme le modèle centralisé, il est basé sur deux types de processeurs : les serveurs et les sources. De la même manière, les processeurs sources envoient des tâches aux processeurs serveurs, qui sont chargés de les calculer. Néanmoins, dans ce modèle les processeurs sont en général sources et serveurs à la fois, en fonction du temps qu'ils choisissent de dédier au calcul et à l'équilibrage de charge. Cette répartition est principalement faite de deux manières :

- **source initiative** : dans cette approche, les sources surchargées transmettent une ou plusieurs tâches aux processeurs serveurs choisie aléatoirement ou après élection (Eager *et al.* 1986). De cette manière les processus serveurs allègent leur charge de travail tandis que les serveurs sont approvisionnés ;
- **serveur initiative** : cette approche est basée sur le principe du « volontariat ». Lorsqu'un serveur est oisif il prend l'initiative de récupérer des tâches depuis les serveurs sources surchargés.

5.1.2.2 Modèles concurrentiels

Une manière de contourner le problème, lié à la décomposition d'une instance, est de dégager du parallélisme d'une autre manière. Au lieu de décomposer le problème en plusieurs sous-problèmes indépendants, une manière simple de produire du parallélisme est de faire travailler de manière concurrentielle plusieurs approches différentes sur une même formule. Ce modèle algorithmique, contrairement au modèle coopérative, ne nécessite pas la mise en place de stratégie d'équilibrage de charges. Néanmoins, ce type d'approche n'est pas toujours évident à mettre en place. En effet, la mise en concurrence de plusieurs méthodes n'a de sens que si les méthodes sont différentes.

Dans le cadre de SAT, cette approche peut facilement être appliquée aux méthodes souffrant du phénomène de longue traînée, telles que la recherche locale et l'approche CDCL. En effet, les algorithmes incomplets de type recherche locale peuvent facilement tirer avantage de l'aspect concurrentiel en lançant plusieurs solveurs avec des graines aléatoires différentes. Quant aux solveurs CDCL, ils peuvent tirer partie d'un lancement en concurrence afin de pallier leur manque de robustesse vis-à-vis du choix des paramètres initiaux.

Dans la suite, nous présentons deux types d'approches basées sur l'un des deux modèles que nous venons d'introduire pour la résolution parallèle du problème SAT.

5.2 Approche parallèle de type chemin de guidage

Dans cette section nous présentons une approche parallèle basée sur le modèle coopératif. Cette approche s'appuie sur une décomposition de l'arbre de recherche à l'aide d'une notion appelée chemin de guidage. Après avoir introduit cette notion nous présentons quelques solveurs qui l'implémentent.

5.2.1 Principe

La méthode de résolution à base de chemin de guidage (*Guiding path*) est l'une des méthodes les plus utilisée pour la résolution parallèle du problème SAT. Cette approche, proposée par Zhang et Bonacina (1994), consiste à conserver des chemins de recherche permettant d'indiquer quels sont les sous-arbres à développer. Un tel chemin est représenté par un ensemble de couples $\{\langle \ell_1, \lambda_1 \rangle, \langle \ell_2, \lambda_2 \rangle, \ldots \langle \ell_n, \lambda_n \rangle\}$, où chaque couple $\langle \ell_i, \lambda_i \rangle$ représente respectivement un littéral de la formule et si l'un des sous-arbres issu de ℓ_i n'est ni en cours de traitement ni traité. Chaque λ_i peut être représenté par une variable booléenne qui est égale à $\bot$ si les deux sous-arbres sont traités et $\top$ sinon.

Exemple 5.2. *Soient Σ une* CNF *tel que $\mathcal{V}_\Sigma = \{x_1, x_2, \ldots, x_n\}$ et $\mathfrak{L} = \{\langle x_1, \top \rangle, \langle x_2, \top \rangle, \langle x_3, \bot \rangle\}$ un chemin de guidage. L'arbre restant à explorer à partir de $\mathfrak{L}$ est donné dans la figure 5.4.*

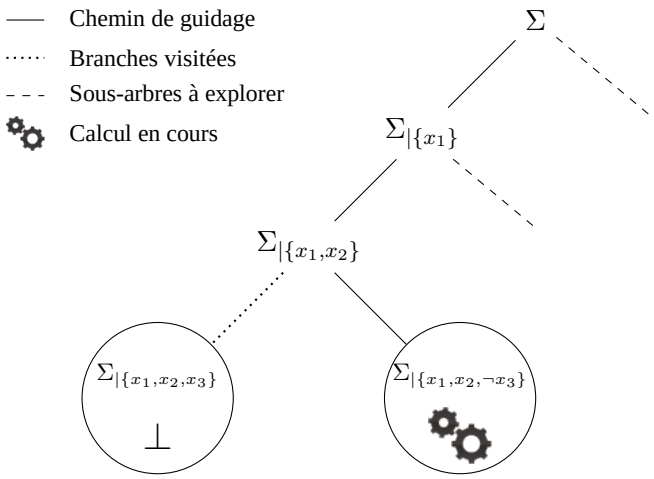


FIGURE 5.4 – Arbre de recherche restant à explorer à partir d'un chemin de guidage.

La résolution d'un problème à l'aide de cette approche consiste alors à résoudre l'ensemble des chemins de guidage. Pour cela, lorsqu'une unité de calcul est oisive, un chemin de guidage lui est assigné par le biais d'un des modèles d'équilibrage de charge présenté précédemment. Lorsque l'ensemble des chemins de guidage a été exploré le solveur peut conclure sur la satisfaisabilité de la formule. Néanmoins, il est possible que certains chemins de guidage soient long à calculer et que d'autres soient très courts. Afin de répartir la tâche équitablement entre les différents processeurs il est possible de décomposer un chemin de guidage en deux sous-chemins de guidage en considérant le littéral complémentaire d'un nœud ouvert. De plus, il est possible d'étendre un chemin de guidage par l'ajout d'un littéral n'apparaissant pas dans celui-ci.

Exemple 5.3. *Considérons le chemin de guidage* $\mathfrak{L} = \{\langle x_1, \top\rangle, \langle x_2, \top\rangle, \langle x_3, \bot\rangle\}$ *de l'exemple 5.2. La figure 5.5 donne une représentation arborescente d'un ensemble de chemins de guidage pouvant être obtenus à partir de* $\mathfrak{L}$*. Sur cette figure chaque nœud représente un chemin de guidage tel que ses fils représentent les sous-chemins obtenus en considérant un nœud ouvert ou un littéral de la formule.*

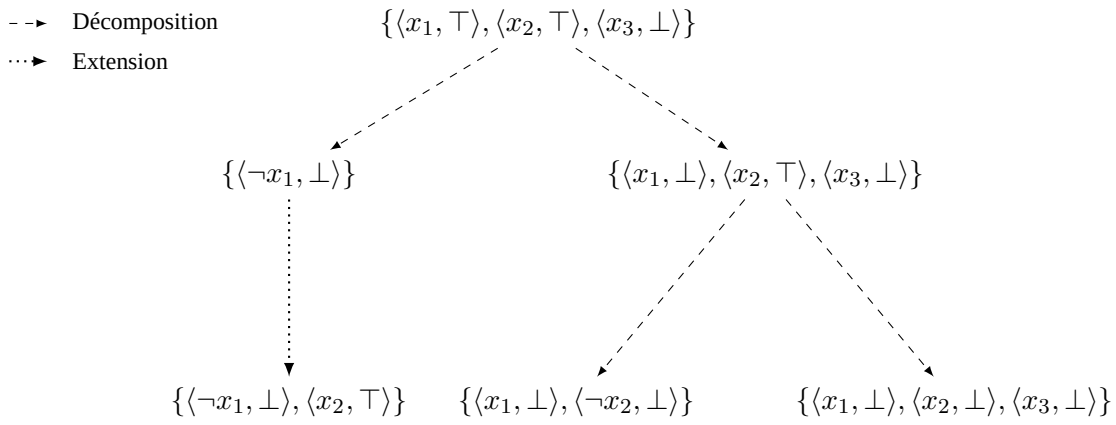


FIGURE 5.5 – Décomposition d'un chemin de guidage.

Du fait de la faible quantité de données à échanger, cette méthode a largement été étudiée en pratique. Nous présentons dans la section suivante quelques solveurs basés sur ce principe.

5.2.2 Présentation de solveurs de type chemin de guidage

Nous donnons ici une liste non exhaustive de solveurs de type *guiding path*.

PSATO Introduit par Zhang *et al.* (1996), il est le premier solveur parallèle a introduire la notion de chemin de guidage. Il est basé sur le solveur séquentiel SATO (*Satisfiability Testing Optimized*) (Zhang 1997) et utilise un modèle centralisé pour partager les tâches. Le processeur maître est chargé de fournir aux processeurs esclaves les sous-arbres qu'ils doivent développer. Afin de toujours pouvoir alimenter les processeurs esclaves, le processeur maître arrête parfois son exécution de manière à créer de nouveaux chemins de guidage. Pour cela, un ou plusieurs processeurs esclaves sont stoppés pour leur demander de retourner leur chemin courant. À partir de ces chemins et de la même manière que celle présentée précédemment, plusieurs nouveaux chemins sont créés.

PASAT Ce solveur est le premier à permettre l'échange de clauses en plus du traditionnel chemin de guidage (Sinz *et al.* 2001, Blochinger *et al.* 2003). Par la force des choses, il a aussi été le premier confronté aux problèmes liés à ce type d'échanges, à savoir le nombre et la taille des clauses à échanger. En effet, les clauses pouvant être apprises étant nombreuses (exponentiel dans le nombre de variables de la formule) et souvent de grande taille, il est nécessaire d'appliquer une politique de discrimination des clauses échangées. Le critère choisi afin de réaliser cela est fonction de la taille de la clause.

Par la suite, plusieurs approches basées sur PASAT ont été proposées. Par exemple, Chrabakh et Wolski (2003) proposent une approche nommée GRADSAT conçue pour être exécutée sur une grille de calcul. GRADSAT utilise un modèle maître/esclave où le maître distribue le travail parmi les esclaves et où chaque esclave correspond à un solveur ZCHAFF (Moskewicz *et al.* 2001a). Les clauses apprises par les esclaves sont gérées par le maître à l'aide d'une base de données distribuée de telle manière que seules les clauses d'une taille inférieure à une certaine constante sont partagées.

PMINISAT Ce solveur, introduit par Chu et Stuckey (2008), est une parallélisation simple de MINISAT 2.0 (Sörensson et Eén 2009) conçu pour être exécuté sur une machine à mémoire partagée. Les chemins de guidage sont conservés au sein d'une base commune, elle même alimentée par des chemins de guidage générés par la plus longue exécution dans l'arbre. Les processus récupèrent un chemin de guidage depuis cette base lorsqu'ils en ont besoin. La particularité de ce solveur est qu'il exploite les connaissances sur les chemins des processus pour améliorer la qualité des clauses échangées. L'idée provient du fait que certaines clauses ne sont utiles que localement dans un sous-arbre. En effet, bien que les clauses puissent être de grande taille, elles peuvent devenir plus petites et par conséquent plus intéressantes après un certain nombre d'affectations. Ce principe permet à PMINISAT d'étendre le partage de clauses du moment que celles-ci deviennent petites dans un certain contexte de recherche.

5.3 Approche de type *portfolio*

Dans cette section nous présentons un type de solveur parallèle s'appuyant sur le modèle concurrentiel. Cette méthode, nommée *portfolio*, consiste à exécuter en parallèle plusieurs solveurs séquentiels. Néanmoins, afin d'être efficace il est nécessaire que les stratégies de recherche utilisées par les différents solveurs forment un ensemble complémentaire et orthogonal.

De plus, certaines approches permettent l'échange d'informations entre les différentes unités de calcul. Ces informations représentent le plus souvent des clauses obtenues par analyse de conflits et permettent d'améliorer les performances du système au-delà de la performance de chaque solveur considéré individuellement.

Bien que « naïve », cette approche est à l'heure actuelle la meilleure manière de résoudre le problème SAT en parallèle. En effet, lors de la compétition SAT 2011, les trois premières places du classement des solveurs parallèles ont été attribuées à des solveurs *portfolio*. Nous présentons en détail deux versions du solveur MANYSAT.

5.3.1 MANYSAT

MANYSAT (Hamadi *et al.* 2009b) est un solveur parallèle de type portfolio incluant toutes les composantes importantes des solveurs SAT modernes (propagation unitaire, *watched litterals*, VSIDS, analyse de conflits, redémarrage, etc.). Ce solveur tente de tirer avantage de la sensibilité aux réglages des paramètres des solveurs CDCL. En effet, changer la politique de redémarrages ou l'heuristique de choix

de variables peut conduire à dégrader ou à améliorer les performances d'un solveur. Partant de ce constat les auteurs proposent d'exécuter plusieurs versions du solveur MINISAT 2.02 (Eén et Sörenson 2004) avec des paramètres différents tout en partageant entres toutes les unités de calcul les clauses apprises. La première place obtenu par MANYSAT dans la catégorie parallèle de la SAT *Race* de 2008 et à la compétition SAT de 2009 montre l'intérêt d'une telle approche. Nous présentons dans la suite les versions 1.1 et 1.5 de ce solveur.

5.3.2 Le solveur MANYSAT 1.1

Le solveur MANYSAT 1.1 (Hamadi *et al.* 2009b) est une approche parallèle de type portfolio élaborée pour 4 unités de calcul. Cet approche consiste à exécuter sur une même instance plusieurs solveurs séquentiels possédant des stratégies différentes. Plus précisément, chaque cœur de calcul représente un solveur CDCL contenant toutes les stratégies classiques. Les solveurs, du fait des différentes stratégies employées, sont alors mis en compétition afin de résoudre le problème initial. Néanmoins, cette compétition est « saine » puisque les différents cœurs partagent des informations entre eux. En effet, lorsqu'un solveur apprend une clause, celle-ci est proposée aux autres unités de calcul. Si cette clause est jugée intéressante (taille inférieure à 8 dans la version originale) pour une unité de calcul, celle-ci est ajoutée à sa base de clauses apprises. Ce partage d'informations permet entre autre aux différents solveurs de coopérer afin de résoudre le problème plus efficacement. La figure 5.6 schématise ce transfert d'informations.

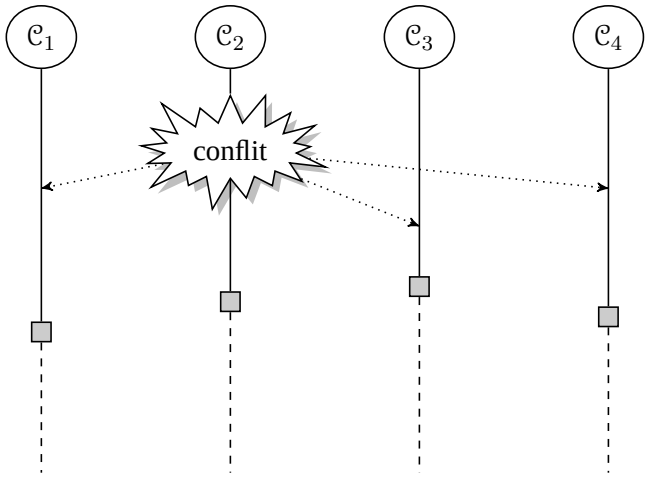


FIGURE 5.6 – Présentation schématique du solveur parallèle MANYSAT. Les lignes continues représentent l'exécution d'un solveur. Les carrés symbolisent les redémarrages tandis que les flèches en pointillés représentent le partage d'informations entre les différents solveurs (ici les clauses apprises).

Afin d'obtenir une approche *portfolio* efficace, il est nécessaire que les différents solveurs mis en concurrence adoptent des stratégies de recherche orthogonales et complémentaires. Afin de réaliser cela, les auteurs proposent de différencier les solveurs sur leurs stratégies de redémarrage, de choix de variables, de choix de polarité et d'apprentissage. Ces différentes stratégies sont détaillées dans le tableau 5.1.

Stratégie	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$
Redémarrage	$dynamic^+$	$dynamic^-$	$geom(100, 1.5)$	$dynamic^+$
Heuristique	VSIDS (98%)	VSIDS (98%)	VSIDS (98%)	VSIDS (97%)
Polarité	*progress saving*	*progress saving*	*false*	*occurence*
Apprentissage	first-UIP	first-UIP	first-UIP étendu	first-UIP
Partage de clauses	taille ≤ 8	taille ≤ 8	taille ≤ 8	taille ≤ 8

TABLE 5.1 – Stratégies des différents solveurs séquentiels utilisés dans le solveur MANYSAT 1.1.

Dans l'optique de ne pas effectuer trop de travail redondant il est souhaitable que les différents solveurs n'explorent pas de la même manière l'espace de recherche. Pour éviter cela il est nécessaire de choisir des stratégies hétérogènes. Néanmoins, l'utilisation d'approches totalement orthogonales peut réduire la pertinence des clauses partagées. En effet, lorsqu'un solveur apprend une nouvelle clause il se trouve dans un certain espace de recherche où celle-ci est utile (propagation du littéral assertif). Lorsque cette clause est partagée avec un autre solveur, il est possible que celui-ci se trouve dans une autre partie de l'espace de recherche ou cette clause n'a aucun intérêt (satisfaite par exemple). Le challenge consiste donc à trouver la bonne « distance » entre les différentes unités de calcul, c'est-à-dire trouver le bon compromis entre intensification et diversification. Pour réaliser cela, les auteurs de MANYSAT proposent une approche de type maître/esclave dans laquelle le maître assure la diversification tandis que l'esclave intensifie certains espaces de recherche fournit par les processus maîtres.

5.3.3 Le solveur MANYSAT 1.5

La version 1.5 du solveur MANYSAT, proposée par Guo *et al.* (2010), est une extension de MANYSAT 1.1 (Hamadi *et al.* 2009b) incorporant une stratégie de type maître/esclave. Dans cette approche le maître est utilisé afin de diversifier la recherche tandis que l'esclave l'intensifie. Plus précisément, chaque maître M possède un esclave E. Lorsque M effectue un redémarrage il transfère à E un ensemble d'informations qu'il doit intensifier. Ces informations consistent en une séquence ordonnée de littéraux collectés par M durant sa dernière analyse de conflits [20]. Le but d'une telle séquence est de forcer E à explorer « différemment » l'espace de recherche autour de la dernière zone de conflit rencontrée par M. En plus de ces informations, les clauses apprises par les différentes unités de calcul sont transmises, de la même manière que dans MANYSAT 1.1, à l'ensemble des solveurs. La figure 5.7 schématise le transfert d'informations entre un maître et son esclave.

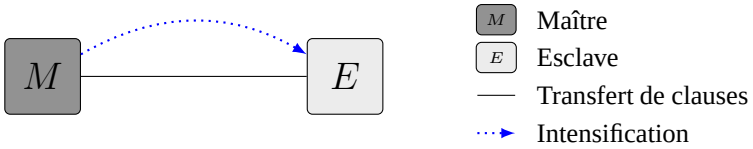


FIGURE 5.7 – Transfert d'informations entre un maître et son esclave.

Afin de caractériser la meilleure configuration à adopter, c'est-à-dire combien de maîtres et surtout combien d'esclaves chaque maître doit gouverner. Les auteurs ont étudié expérimentalement l'ensemble des combinaisons possibles pour une architecture comportant 4 unités de calcul. La conclusion de ces expérimentations conduit à considérer une topologie constituée de 2 maîtres, chacun possédant 1 esclave.

20. Ces littéraux sont ceux utilisés lors de la génération de la clause assertive (voir 3.1.5.1).

La figure 5.8 schématise une telle topologie ainsi que les échanges d'informations entre les différents solveurs.

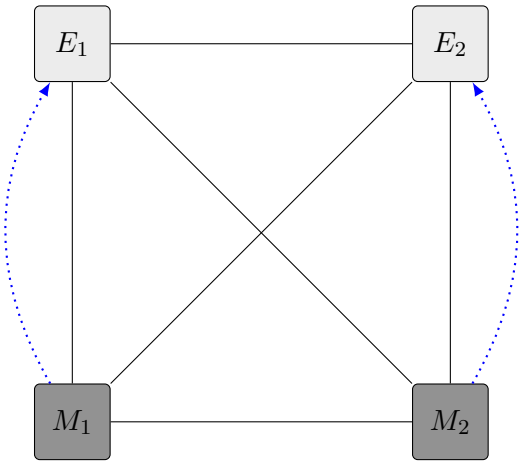


FIGURE 5.8 – Topologie Maître/Esclave.

Bien que la version 1.5 du solveur MANYSAT soit basée sur la version 1.1, les stratégies utilisées pour chacune des unités de calcul sont légèrement différentes. En effet, afin de diversifier correctement l'espace de recherche les auteurs ont opté pour des politiques de redémarrages très agressives, et un schéma d'apprentissage identique pour l'ensemble des cœurs. Le tableau 5.2 récapitule les stratégies utilisées par les différentes unités de calcul.

Stratégie	M_1	M_2	E_1	E_2
Redémarrage	$luby(512)$	$dynamic^+$	Contrôlé par M_1	Contrôlé par M_2
Heuristique	VSIDS (98%)	VSIDS (98%)	Obtenue via M_1	Obtenue via M_2
Polarité	*progress saving*	*progress saving*	*progress saving*	*false*
Apprentissage	first-UIP étendu	first-UIP étendu	first-UIP étendu	first-UIP étendu
Partage de clauses	taille ≤ 8	taille ≤ 8	taille ≤ 8	taille ≤ 8

TABLE 5.2 – Stratégies des différents solveurs séquentiels utilisés dans le solveur MANYSAT 1.5.

Les auteurs ont montré à l'aide de cette approche que le processus d'intensification et de diversification était un élément important des solveurs parallèles de type *portfolio*. Partant de ce constat, nous proposons, dans le chapitre 10, une approche permettant d'estimer la distance entre deux unités de calcul. Cette distance nous indique si deux solveurs sont en train d'explorer le même espace de recherche. Lorsque deux solveurs sont considérés comme trop proches, nous modifions les paramètres d'un des solveurs afin de diversifier son espace de recherche de manière à les éloigner.

5.4 Conclusion

Dans ce chapitre, nous avons présenté deux modèles algorithmiques pour la résolution de problèmes sur une architecture multi-cœurs. Dans le cadre de SAT ces deux paradigmes de résolution ont conduit à l'élaboration de deux schémas que sont : les chemin de guidage et les approches *portfolio*. Les chemins de guidage bien qu'étant largement étudiés sont à l'heure actuelle moins performants que les approches

portfolio. En particulier, un solveur tel que MANYSAT a permis de montrer que l'utilisation d'approches coopératives et orthogonales est nettement plus appropriée dans le contexte de solveurs SAT modernes.

Conclusion de l'étude bibliographique

Du fait de leur appartenance à la classe de complexité NP-complet, les problèmes SAT et CSP sont très difficiles à résoudre. Cette difficulté a mené à l'élaboration de différentes méthodes de résolution. Parmi ces dernières, deux types d'approches se sont principalement dégagés : la recherche locale et les méthodes de type *backtracking*. Nous avons vu que ces deux paradigmes de résolution ont chacun des mécanismes permettant d'obtenir de bons résultats sur certaines classes d'instances et que leur combinaison au sein d'une approche hybride peut permettre de les cumuler. De telles combinaisons, bien qu'en théorie intéressantes, se sont révélées être en pratique moins efficaces que chacune des approches prisent séparément. Néanmoins, nous pouvons voir au travers de deux de nos contributions que de tels schémas algorithmiques ne sont pas dénués d'intérêt. En effet dans la partie II, nous présentons deux nouvelles approches hybrides pour la résolution des problèmes SAT et CSP très robustes, c'est-à-dire permettant de résoudre globalement plus d'instances que chacune des approches prises séparément. La première, présentée dans le chapitre 6, consiste à étendre le schéma d'hybridation proposé par Cha et Iwama (1996) à l'aide d'une approche s'appuyant sur la construction d'un graphe d'implications afin d'extraire des clauses par l'application d'un chemin de résolutions. La seconde contribution consiste en un nouveau schéma d'hybridation pouvant être aussi bien appliqué au cadre SAT (chapitre 7) qu'au cadre CSP (chapitre 8).

Les solveurs actuels sont souvent composés de plusieurs éléments. Cette particularité fait qu'améliorer l'efficacité d'une approche peut se faire de différentes manières (structures de données efficaces, stratégies de redémarrages, heuristiques de choix de variables ...). Dans la partie III nous proposons deux contributions indépendantes pour améliorer la résolution pratique du problème SAT. Dans un premier temps (chapitre 9) nous présentons une nouvelle mesure permettant d'estimer la pertinence d'une clause apprise. Une fois cette dernière définie, nous proposons de l'utiliser au sein d'un nouveau schéma de nettoyage de la base de clauses apprises. Pour terminer, nous présentons dans le chapitre 10 une approche permettant d'ajuster de manière dynamique les différentes heuristiques de choix de polarité utilisées par les unités de calcul d'un solveur parallèle de type *portfolio*.

Deuxième partie
Méthodes de résolution hybride

Introduction

Comme nous avons pu le voir dans le chapitre 4, la plupart des algorithmes de recherche hybride utilisent principalement une recherche en profondeur d'abord avec retour arrière (voir chapitre 3) comme moteur principal. De plus, les informations sont uniquement transmises de la méthode secondaire vers la méthode principale. Ceci est dommageable, car chaque souche (complète ou non) peut donner des informations qui seront utiles à l'autre pour la suite de la recherche. C'est ce que nous nous sommes employés à faire dans les différents algorithmes présentés dans cette partie.

Cette partie est organisée de la manière suivante. Dans le premier chapitre, une nouvelle approche hybride pour la résolution de SAT est proposée (Audemard *et al.* 2009a;b). Cette méthode consiste à étendre l'analyse de conflits, utilisée dans les solveurs les plus performants à l'heure actuelle, à la recherche locale. Cette analyse a pour objectif de comprendre les raisons pour lesquelles un minimum local a été atteint et ainsi permettre de mieux s'en échapper. De plus, nous montrons que sous certaines conditions cette analyse permet de prouver l'inconsistance d'une formule.

Les deux derniers chapitres concernent l'application d'un nouveau schéma d'hybridation pour la résolution des problèmes SAT et CSP (Audemard *et al.* 2009c; 2010a;c;b, Grégoire *et al.* 2011a;b). Il comprend une combinaison synergique de la recherche locale et d'éléments de techniques complètes, qui souvent surpassent les approches complètes usuelles. Cette méthode n'est pas seulement complète : elle est aussi robuste puisqu'elle peut aussi bien résoudre les instances satisfiables qu'insatisfiables, structurées ou aléatoires. En effet notre étude expérimentale, conduite dans le cadre SAT et dans le cadre CSP, montre qu'elle résout globalement plus d'instances que les techniques couramment utilisées.

Chapitre 6

Analyse de conflits et recherche locale

Sommaire

DANS CE CHAPITRE, nous présentons une nouvelle approche pour sortir des minima locaux dans le cadre de la recherche locale. Cette approche est basée sur le principe d'analyse de conflits utilisé dans les solveurs SAT modernes. Nous proposons une extension du graphe d'implications (voir section 3.1.5.1) au cadre de la recherche locale où plusieurs conflits sont présents pour une interprétation donnée. Nous présentons ensuite deux approches permettant de construire et d'exploiter de tels graphes. Enfin, nous étendons le schéma classique de WSAT pour y intégrer notre analyse de conflits. Les résultats expérimentaux montrent que l'intégration de notre système d'analyse de conflits améliore sensiblement les performances de WSAT sur les problèmes structurés. De plus cette méthode isolant des sous-problèmes inconsistants est capable de montrer que l'instance n'admet pas de modèle.

Ces travaux ont donné lieu à deux publications (Audemard *et al.* 2009a;c).

6.1 Analyse de conflits et recherche locale

Nous avons vu dans le chapitre 3.1.5.1 que la construction et l'interprétation du graphe d'implications était définie en fonction de la propagation unitaire. Ici étant donné que nous travaillons dans le cadre de la recherche locale, la propagation unitaire est inexistante. Pour remédier à ce problème nous avons changé le mode de construction du graphe d'implications et proposons une nouvelle interprétation de celui-ci permettant l'extraction de clauses conflits.

6.1.1 Définition des graphes conflits

Nous commençons par définir ce qu'est un graphe conflit pour une interprétation complète. Dans ce dernier, les conflits sont naturellement représentés par les clauses falsifiées par l'interprétation complète tandis que les clauses unisatisfaites sont utilisées pour remplacer la notion de propagation unitaire.

Définition 6.1 (Graphe conflit sur z)**.** *Soient Σ une formule sous forme* CNF*, $\mathcal{I}$ une interprétation complète conflictuelle de Σ. Étant données deux clauses de Σ, $\beta = \{\beta_1, \ldots, \beta_k, z\}$ falsifiée par $\mathcal{I}$ et $\gamma = \{\gamma_1, \ldots, \gamma_l, \overline{z}\}$ unisatisfaite en z pour $\mathcal{I}$, nous construisons le graphe conflit noté $\mathcal{G}^z_{\Sigma,\mathcal{I}} = (\mathcal{N}, \mathcal{A})$ de la manière suivante :*

1. – $\{z, \overline{z}, \bot\} \subseteq \mathcal{N}$;
   – $\{\overline{\gamma_1}, \ldots, \overline{\gamma_l}\} \subseteq \mathcal{N}$;
   – $\{\overline{\beta_1}, \ldots, \overline{\beta_k}\} \subseteq \mathcal{N}$;
2. – $\{(z, \bot), (\overline{z}, \bot)\} \subseteq \mathcal{A}$;
   – $\{(\overline{\beta_1}, z), \ldots, (\overline{\beta_k}, z)\} \subseteq \mathcal{A}$;
   – $\{(\overline{\gamma_1}, \overline{z}), \ldots, (\overline{\gamma_k}, \overline{z})\} \subseteq \mathcal{A}$;
3. $\forall x \in \mathcal{N}$, *si* $x \neq z$ *et* $\alpha = \bigwedge\{y \in \mathcal{N} \mid (y, x) \in \mathcal{A}\} \not\models \bot$ *alors* $\overline{\alpha} \vee x \in \Sigma$ *et est unisatisfaite en* x.

Pour un littéral donné z, différents choix sont possibles pour choisir les clauses β et γ. De plus, il est possible que plusieurs clauses unisatisfaites expliquent un même littéral, et donc en règle générale, il n'y a pas unicité du graphe conflit.

Exemple 6.1. *Considérons la formule* CNF $\Sigma = \{\phi_1, ..., \phi_{11}\}$ *telle que :*

$\phi_1 : (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ $\quad \phi_2 : (x_1 \vee \neg x_4 \vee \neg x_5)$ $\quad \phi_3 : (x_2 \vee \neg x_1)$ $\quad \phi_4 : (x_4 \vee \neg x_7 \vee \neg x_6)$
$\phi_5 : (x_3 \vee \neg x_5)$ $\quad \phi_6 : (x_5 \vee \neg x_7)$ $\quad \phi_7 : (x_6 \vee \neg x_8)$ $\quad \phi_8 : (x_7 \vee \neg x_8)$
$\phi_9 : (x_8 \vee \neg x_4)$ $\quad \phi_{10} : (x_1 \vee \neg x_8)$ $\quad \phi_{11} : (x_7 \vee \neg x_9)$

Soit l'interprétation complète $\mathcal{I}$ construite sur $\mathcal{V}_\Sigma$ telle que $\mathcal{I}(x_i) = \top$ pour tout $x_i \in \mathcal{V}_\Sigma$. La figure 6.1 schématise un graphe conflit $\mathcal{G}^{x_1}_{\Sigma,\mathcal{I}}$ construit sur la variable x_1.

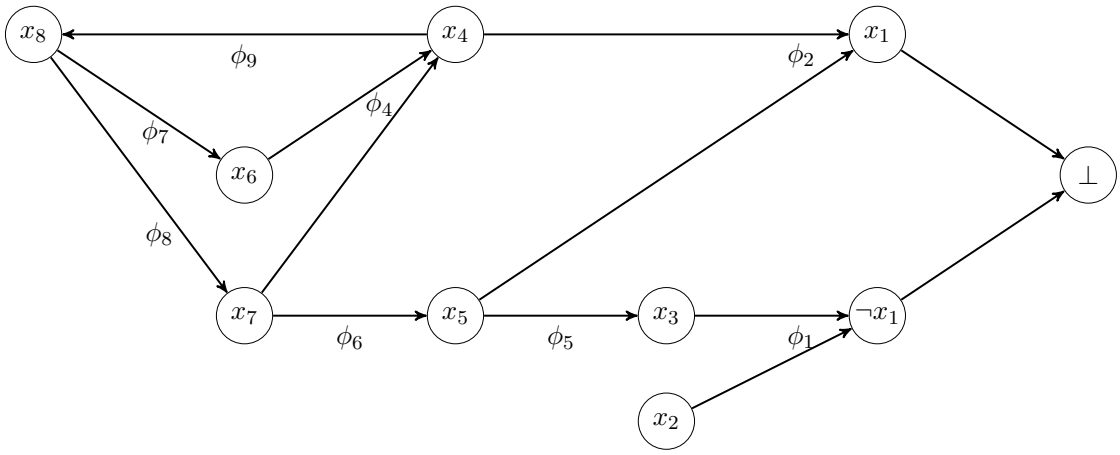


FIGURE 6.1 – Graphe conflit défini sur la variable x_1.

Il est important de noter que l'existence d'un tel graphe n'est pas toujours assurée, et la proposition suivante exprime les conditions sous lesquelles la construction de ce graphe conflit est possible.

Proposition 6.1. *Soient Σ une formule sous forme CNF, $\mathcal{I}$ une interprétation complète construite sur les variables de Σ. Le graphe conflit $\mathcal{G}^x_{\Sigma,\mathcal{I}}$ est constructible si et seulement si x et $\neg x$ apparaissent respectivement dans une clause fausse et une clause unisatisfaite.*

Preuve 6.1. *Cette proposition découle directement de la définition d'un graphe conflit.*

Corrolaire 6.2. *Soit $\alpha \in \Sigma$ une clause critique (voir définition 1.42). $\forall x \in \alpha$ il est possible de construire un graphe conflit défini sur x.*

Preuve 6.2. *Par définition d'une clause critique $\alpha \in \Sigma$ nous avons $\forall x \in \alpha$, $\exists \beta \in \Sigma$ unisatisfaite en x. D'après la proposition 6.1, il est facile de voir que $\forall x \in \alpha$ il existe un graphe conflit en x.*

Lorsque nous nous trouvons dans un minimum local, toutes les clauses non satisfaites sont critiques. Le corrolaire 6.2 nous assure donc que, dans ce cas, il est toujours possible de créer un graphe conflit. Ceci est très important, puisque cela va nous permettre de construire un graphe afin de générer un *nogood*. Néanmoins, générer de tels *nogoods* en partant du graphe conflit est un problème relativement difficile. En effet, la notion de niveau étant absente, l'analyse de conflit classique basée sur la notion d'UIPs ne peut être étendue. De plus, les graphes conflits, tels que définis précédemment, peuvent contenir des cycles. Pour palier à ces problèmes, nous proposons dans la suite deux méthodes basées respectivement sur la notion de chemin critique et sur la notion d'interprétation partielle dérivée.

6.1.2 Construction et analyse d'un graphe conflit basée sur la notion de chemin critique

Afin de palier les problèmes énoncés précédemment, nous proposons une première méthode basée sur la notion de chemin critique [21]. Étant donnée une interprétation conflictuelle, une séquence de clauses est extraite de la manière suivante.

Définition 6.2 (chemin critique)**.** *Soient Σ une formule* CNF *et ϕ une clause de Σ falsifiée par l'interprétation complète $\mathcal{I}$, le chemin critique $< \sigma_0, ..., \sigma_k >$ est une séquence de clauses de Σ construite de manière récursive de la façon suivante :*

1. *$\sigma_0 = \phi$ et $\mathcal{E}_0 = \emptyset$;*
2. *$\forall i > 0, \sigma_i = x \vee \gamma_i$ telle que :*
   - *σ_i est une clause de Σ unisatisfaite en x par l'interprétation $\mathcal{I}$;*
   - *$\neg x \in \sigma_{i-1}$, $\neg x \notin \mathcal{E}_{i-1}$ et $\mathcal{E}_i = \mathcal{E}_{i-1} \cup \{\neg x\}$;*
3. *σ_k est telle que le point précédent ne peut être appliqué.*

Exemple 6.2. *Considérons la formule Σ de l'exemple 6.1 et l'interprétation complète $\mathcal{I}$ qui assigne à vrai l'ensemble des variables de Σ, la clause ϕ_1 est falsifiée par l'interprétation complète $\mathcal{I}$, nous pouvons construire le chemin critique suivant : $< \phi_1, \phi_3, \phi_2, \phi_5, \phi_6, \phi_4, \phi_7, \phi_8, \phi_9 >$.*

Dans cette méthode la construction du graphe conflit est implicite. Étant donné un chemin critique $\mathfrak{C} =< \sigma_0, \ldots, \sigma_k >$ construit à partir d'une formule Σ et une interprétation complète conflictuelle $\mathcal{I}$, le graphe conflit $\mathcal{G}^z_{\Sigma,\mathcal{I}} = (\mathcal{N}, \mathcal{A})$ correspondant est tel que :

- $\mathcal{N} = \{\bot, \neg z\} \cup \{\ell \in \mathcal{I} | \exists \sigma \in \mathfrak{C}$ tel que $\neg \ell \in \sigma\}$;
- $\mathcal{A} = \{(z, \bot), (\neg z, \bot)\} \cup \{(\ell, z) | \sigma_0 = \beta \vee z$ et $\neg \ell \in \beta\} \cup \bigcup_{i=1}^{n} \{(x, y) | \sigma_i = \beta_i \vee y$ et $\neg x \in \beta_i\}$.

Exemple 6.3. *La figure 6.2 illustre le graphe conflit associé au chemin critique de l'exemple 6.2.*

21. Cette notion de chemin critique est différente de la notion de chemin critique usuel défini en théorie des graphes. Le terme critique est repris ici en référence aux clauses critiques

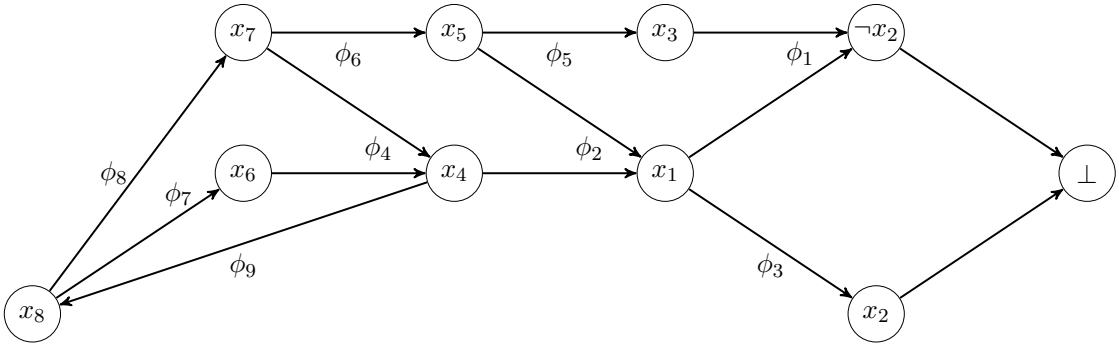


FIGURE 6.2 – Graphe conflit défini à partir d'un chemin critique.

L'algorithme 6.1 permet la construction d'un chemin critique en fonction d'une formule Σ et d'une interprétation complète conflictuelle $\mathcal{I}$. De la même manière que pour la construction d'un graphe conflit, il est possible de construire plusieurs chemins critiques à partir d'une même interprétation conflictuelle. Les graphes pouvant être obtenus par le biais de cette méthode sont intrinsèquement liés aux choix des clauses effectués aux lignes 2 et 5. Lors de nos différentes études expérimentales, reportées dans la section 6.2.2.1, nous avons choisi de sélectionner ces dernières de manière aléatoire.

Algorithme 6.1 : Extraction chemin critique

Données : Σ une formule et $\mathcal{I}$ une interprétation complète conflictuelle
Résultat : $\mathfrak{C}$ une séquence de clauses

1 **Début**
2 $\quad \mathfrak{C} \longleftarrow \alpha$ telle que $\alpha \in \Sigma$ et $\mathcal{I} \not\models \alpha$;
3 $\quad \mathcal{E} \longleftarrow \emptyset$; `/* littéraux déjà considérés */`
4 $\quad$ **tant que** $\exists (x \vee \beta) \in \mathfrak{C}$ *telle que* $\neg x \notin \mathcal{E}$ **faire**
5 $\quad\quad \mathfrak{C} \longleftarrow \mathfrak{C} \cup \{\beta\}$ telle que $\beta \in \Sigma$ et $|\mathcal{I} \cap \beta| = 1$;
6 $\quad\quad \mathcal{E} \longleftarrow \mathcal{E} \cup \{\neg x\}$;
7 $\quad$ **retourner** $\mathfrak{C}$;
8 **Fin**

À partir d'un chemin critique généré à l'aide de la procédure précédente, l'analyse de conflit est obtenue en effectuant une succession de résolution prenant en compte les clauses du chemin critique dans l'ordre. Cet ensemble de clauses est obtenu formellement comme suit :

Définition 6.3 (chemin de résolution critique)**.** *Soit* $\mathfrak{C} =< \sigma_0, \ldots, \sigma_k >$ *un chemin critique, l'ensemble des nogoods* $\{\nu_0, \nu_1, \ldots, \nu_k\}$ *pouvant être généré à partir du chemin critique* $\mathfrak{C}$ *est défini récursivement de la manière suivante :*

- $\nu_0 = \sigma_0$ *;*
- $\nu_i = \eta[x, \nu_{i-1}, \sigma_i]$ *telle que* $0 < i \leq k$, $x \in \nu_{i-1}$ *et* $\neg x \in \sigma_i$.

Exemple 6.4. *Considérons le chemin critique* $< \phi_1, \phi_3, \phi_2, \phi_5, \phi_6, \phi_4, \phi_7, \phi_8, \phi_9 >$ *de l'exemple 6.2, l'ensemble de nogoods pouvant être généré est le suivant :*

$\nu_0 = \phi_1 = \neg x_1 \vee \neg x_2 \vee \neg x_3$
$\nu_1 = \eta[x_2, \nu_0, \phi_3] = \neg x_1 \vee \neg x_3$
$\nu_2 = \eta[x_1, \nu_1, \phi_2] = \neg x_3 \vee \neg x_4 \vee \neg x_5$
$\nu_3 = \eta[x_3, \nu_2, \phi_5] = \neg x_4 \vee \neg x_5$

$\nu_4 = \eta[x_5, \nu_3, \phi_6] = \neg x_4 \vee \neg x_7$
$\nu_5 = \eta[x_4, \nu_4, \phi_4] = \neg x_6 \vee \neg x_7$
$\nu_6 = \eta[x_6, \nu_5, \phi_7] = \neg x_7 \vee \neg x_8$
$\nu_7 = \eta[x_7, \nu_6, \phi_8] = \neg x_8$
$\nu_8 = \eta[x_8, \nu_7, \phi_9] = \neg x_4$

L'algorithme 6.2 permet d'analyser le graphe conflit grâce à la méthode décrite précédemment. À partir d'une formule Σ et d'une interprétation conflictuelle $\mathcal{I}$, cette méthode commence par générer le chemin critique $\mathfrak{C} =< \sigma_0, ..., \sigma_k >$ (ligne 3). Ce chemin critique est ensuite utilisé afin d'extraire un *nogood* du graphe conflit (lignes 4 à 7). Cette clause est la dernière clause générée lors du processus de résolution. Pour terminer, cette procédure inverse la valeur de vérité d'une variable de $\mathcal{I}$ afin de satisfaire la raison de l'échec qu'elle vient de calculer (ligne 9).

Algorithme 6.2 : Extraction d'un *nogoods* d'un chemin critique

Données : Σ une formule et $\mathcal{I}$ une interprétation complète conflictuelle se trouvant dans un minimum local
Résultat : β une clause impliquée par Σ et l'interprétation $\mathcal{I}$ modifiée

1 **Début**
2 $i \longleftarrow 0$;
3 $[\sigma_0, \sigma_1, \ldots, \sigma_k] \longleftarrow$`extractionCheminCritique(`$\Sigma, \mathcal{I}$`);`
4 $\beta \longleftarrow \sigma_0$; `/* résultat de la` $i^{\text{ème}}$ `résolution */`
5 **tant que** $(i \leqslant k)$ **faire**
6 $i \longleftarrow i + 1$;
7 $\beta \longleftarrow \eta[x, \beta, \sigma_i = (x \vee \tau)]$;
8 $\ell \in \beta$;
9 $\mathcal{I} \longleftarrow \mathcal{I} \setminus \{\neg\ell\} \cup \{\ell\}$;
10 **retourner** β;
11 **Fin**

6.1.3 Construction et analyse d'un graphe conflit basée sur la notion d'interprétation partielle dérivée

La seconde méthode proposée est basée sur la propagation unitaire. Partant de l'interprétation complète conflictuelle $\mathcal{I}$, nous allons construire une interprétation partielle par propagation unitaire dont les points de choix ont les mêmes valeurs de vérité dans les deux interprétations. Cette interprétation, que nous définissons ci-dessous, nous permet ensuite de définir notre graphe conflit acyclique.

Définition 6.4 (interprétation partielle dérivée)**.** *Soient Σ une formule sous forme* CNF*, $\mathcal{I}$ une interprétation complète conflictuelle construite sur $\mathcal{V}_\Sigma$, $\delta = [x_1, x_2, \ldots, x_n]$ une séquence de décisions telle que $\delta \subseteq \mathcal{I}$ et $\mathcal{H} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ la pile de propagations obtenue à partir de Σ et de δ. L'interprétation partielle dérivée $\mathcal{I}' = \bigcup_{i=1}^{l-1} \mathcal{S}_i \cup \{(x_l), x_l^1, x_l^2, \ldots, x_l^m\}$ telle que $\bigcup_{i=1}^{l-1} \mathcal{S}_i \subseteq \mathcal{I}$, $\mathcal{S}_l \not\subseteq \mathcal{I}$ et $\langle (x_l), x_l^1, x_l^2, \ldots, x_l^m \rangle$ est une sous séquence de $\mathcal{S}_l$ telle que $x_l^m \notin \mathcal{I}$ et $\forall j, 1 \leqslant j < m, x_l^j \in \mathcal{I}$.*

Notation 6.1. *L'ensemble des points de choix $[x_1, x_2, \ldots, x_l]$ est nommé ensemble conflit et nous appelons littéral conflit le littéral $x \in \mathcal{I}' \setminus \mathcal{I}$, c'est-à-dire x_l^m.*

Exemple 6.5. *Reprenons la formule Σ de l'exemple 6.1 et l'interprétation complète conflictuelle $\mathcal{I}$ qui associe vrai à toutes les variables de Σ. Considérons comme séquence de décision $\delta = [x_1, x_2, \ldots, x_9]$ la séquence obtenue en considérant l'ensemble des littéraux de $\mathcal{I}$ dans l'ordre lexicographique. Nous obtenons l'interprétation partielle dérivée suivante :*

- $\mathcal{I}'_0 = \emptyset$
- $\mathcal{I}'_1 = \{(x_1), x_2^1, \neg x_3^1\}$

La littéral conflit est alors $\neg x_3$ et l'ensemble conflit est limité $\{x_1\}$.

Bien entendu, le choix des variables apparaissant dans l'ensemble conflit est un choix heuristique et conduit, comme le montre l'exemple suivant, à différentes interprétations partielles dérivées.

Exemple 6.6. *Soient Σ le formule de l'exemple 6.1 et l'interprétation complète conflictuelle $\mathcal{I}$ qui associe vrai à toutes les variables de Σ. Considérons comme séquence de décision $\delta = [x_9, x_8, \ldots, x_1]$ la séquence obtenue en considérant l'ensemble des littéraux de $\mathcal{I}$ dans l'ordre lexicographique inverse. Nous obtenons l'interprétation partielle dérivée suivante :*

- $\mathcal{I}'_0 = \emptyset$
- $\mathcal{I}'_1 = \{(x_9^1), x_7^1, x_5^1, x_1^3\}$
- $\mathcal{I}'_2 = \{(x_9^1), x_7^1, x_5^1, x_1^3, (x_8^2), x_6^2, x_4^2, x_1^2, x_2^2, \neg x_2^2\}$

Le littéral conflit est x_2 et l'ensemble conflit est $\{x_9, x_8\}$.

L'interprétation complète étant conflictuelle, son interprétation partielle dérivée va différer sur au moins un littéral (le littéral conflit). La proposition suivante exprime le fait qu'il existe au moins une clause falsifiée par l'interprétation complète conflictuelle contenant ce littéral. C'est à partir de celle-ci que le graphe conflit est construit.

Proposition 6.3. *Soient Σ une formule* CNF*, $\mathcal{I}$ une interprétation complète conflictuelle de Σ et $\mathcal{I}'$ une interprétation partielle dérivée de $\mathcal{I}$. Soit x le littéral conflit, alors $exp(x) \subseteq \mathcal{I}$ et la clause $\overrightarrow{cla}(x)$ est falsifiée par l'interprétation $\mathcal{I}$.*

Preuve 6.3. *Tout d'abord il est facile de voir que par construction de $\mathcal{I}'$ nous avons $exp(x) \subseteq \mathcal{I}$. En effet, supposons que $exp(x) \not\subseteq \mathcal{I}$ alors $\exists y \in exp(x)$ telle que $y \notin \mathcal{I}$. Par définition $exp(x) \subseteq \mathcal{I}'$, donc $y \in \mathcal{I}'$ et par conséquent le littéral y est aussi un littéral conflit. Par construction de $\mathcal{I}'$ il ne peut y avoir qu'un littéral conflit, nous avons alors $y = x$. Absurde, une variable propagée ne peut pas appartenir à son explication.*

Montrons à présent que l'interprétation $\mathcal{I}$ falsifie la clause $\overrightarrow{cla}(x)$. Pour cela, raisonnons également par l'absurde et supposons $\overrightarrow{cla}(x)$ est satisfaite par $\mathcal{I}$. En premier lieu, nous pouvons noter que x est une variable obtenue par propagation unitaire. Il est donc possible de trouver une explication $exp(x)$ et une clause $\overrightarrow{cla}(x) \in \Sigma$ telle que $\overrightarrow{cla}(x) = \overline{exp(x)} \vee x$. Nous savons par ailleurs que $exp(x) \subseteq \mathcal{I}$, donc $\mathcal{I} \not\models \overline{exp(x)}$. Puisque $\overrightarrow{cla}(x)$ est satisfaite sous l'interprétation $\mathcal{I}$ elle ne peut l'être qu'en x. Ce qui est absurde puisque $x \notin \mathcal{I}$.

La proposition suivante exprime le fait que toutes les clauses (hormis la clause falsifiée) ayant servi à la construction du graphe sont unisatisfaites.

Proposition 6.4. *Soient Σ une formule* CNF*, $\mathcal{I}$ une interprétation complète conflictuelle de Σ, $\mathcal{I}'$ une interprétation partielle dérivée de $\mathcal{I}$ en fonction de Σ et x le littéral conflit associé à $\mathcal{I}'$. Considérons le graphe d'implications $\mathcal{G}_\Sigma^{\mathcal{I}'} = (\mathcal{N}, \mathcal{A})$, alors $\forall y \in \mathcal{N} \setminus \{x\}$ nous avons $\overrightarrow{cla}(y) = \bot$ ou $\overrightarrow{cla}(y)$ est unisatisfaite par $\mathcal{I}$ en x.*

Preuve 6.4. *Deux cas sont à considérer :*

1. *y est un point de choix et alors $\overrightarrow{cla}(y) = \bot$;*
2. *y n'est pas un littéral propagé. Il existe $\overrightarrow{cla}(y) \in \Sigma$ telle que $\overrightarrow{cla}(y) = \overline{exp(y)} \vee y$. Par construction de $\mathcal{I}'$, nous avons $x \notin exp(y)$ et $\mathcal{I}' \setminus \{x\} \subset \mathcal{I}$. Puisque $y \neq x$ nous avons $exp(y) \cup \{y\} \subseteq \mathcal{I}'$. Par transitivité nous avons $exp(y) \cup \{y\} \subseteq \mathcal{I} \setminus \{x\}$. Donc la clause $\overrightarrow{cla}(y)$ est unisatisfaite par $\mathcal{I}$ en y.*

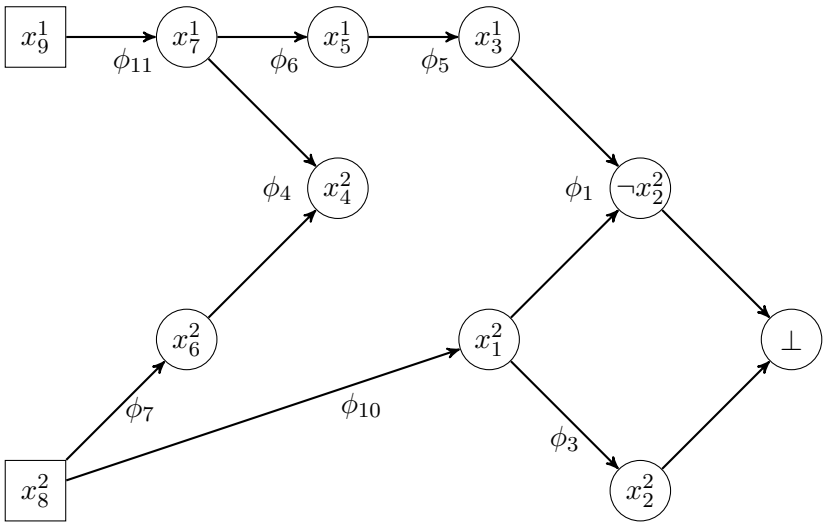
Nous prouvons dans la proposition 6.5 que le graphe d'implications obtenu à partir de l'interprétation partielle dérivée peut être étendu en un graphe conflit sur le littéral conflit. Ayant maintenant un graphe d'implications comme ceux utilisés dans les solveurs CDCL classiques (Moskewicz *et al.* 2001a), nous allons pouvoir analyser celui-ci pour générer un nogood.

Proposition 6.5. *Soient Σ une formule* CNF*, $\mathcal{I}$ une interprétation complète conflictuelle de Σ, $\mathcal{I}'$ une interprétation partielle dérivée et x le littéral conflit associé à $\mathcal{I}'$. Si $\exists \alpha \in \Sigma$ unisatisfaite par $\mathcal{I}$ en x, alors il est possible d'étendre le graphe d'implication $\mathcal{G}_\Sigma^{\mathcal{I}'} = (\mathcal{N}, \mathcal{A})$ associé à $\mathcal{I}'$ en un graphe conflit $\mathcal{G}_{\mathcal{I},\Sigma}^{x} = (\mathcal{N}', \mathcal{A}')$ de la manière suivante :*
- *$\mathcal{N}' = \mathcal{N} \cup \{y \in \overline{\alpha} \setminus x\} \cup \{\overline{x}, \bot\}$;*
- *$\mathcal{A}' = \mathcal{A} \cup \{(y, \overline{x}) | y \in \overline{\alpha} \setminus x\} \cup \{(x, \bot), (\overline{x}, \bot)\}$.*

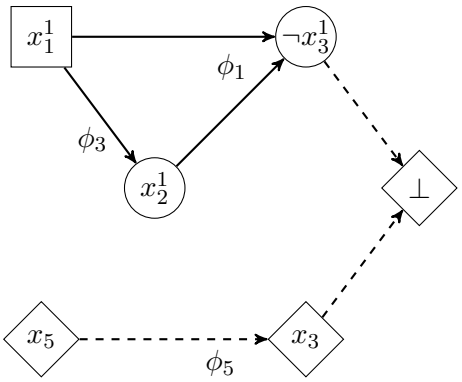
Preuve 6.5. *Avant de vérifier que $\mathcal{G}_{\mathcal{I},\Sigma}^{x}$ est bien un graphe conflit valide, il faut identifier les clauses $\beta = \{x_1 \vee x_2 \vee \cdots \vee x_k, z\} \in \Sigma$ falsifiée par $\mathcal{I}$ et $\gamma = \{y_1 \vee y_2 \vee \ldots y_l \vee \overline{z}\} \in \Sigma$ unisatisfaite en z pour l'interprétation $\mathcal{I}$. Par hypothèse, la clause α est unisatisfaite par $\mathcal{I}$ en x. Nous pouvons donc poser $\gamma = \alpha$. D'après la proposition 6.3, nous pouvons prendre pour β la clause $\overrightarrow{cla}(x)$. En effet, $x \in \overrightarrow{cla}(x)$ et la clause $\overrightarrow{cla}(x)$ est bien falsifiée par $\mathcal{I}$. Les deux clauses servant à la construction du graphe conflit étant identifiées, pour valider que $\mathcal{G}_{\mathcal{I},\Sigma}^{x} = (\mathcal{N}', \mathcal{A}')$ est un graphe conflit, il suffit, d'après la définition 6.1, de vérifier les propriétés suivantes :*

1. – *$\{x, \overline{x}, \bot\} \subseteq \mathcal{N}'$. Par hypothèse, nous avons $\{\overline{x}, \bot\} \subseteq \mathcal{N}'$, reste à montrer que $x \in \mathcal{N}'$. Nous savons que $x \in \mathcal{I}'$, par construction du graphe $\mathcal{G}_\Sigma^{\mathcal{I}'}$, nous avons $x \subseteq \mathcal{N}$. Par conséquent, puisque $\mathcal{N} \subseteq \mathcal{N}'$, $x \in \mathcal{N}'$;*
   – *$\{\overline{y_1}, \overline{y_2}, \ldots, \overline{y_l}\} \subseteq \mathcal{N}'$. Nous avons $\{y \in \overline{\alpha} \setminus x\} \subseteq \mathcal{N}'$ et $\gamma = \alpha$. Donc $\{y \in \overline{\gamma} \setminus x\} \subseteq \mathcal{N}'$;*
   – *$\{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k}\} \subseteq \mathcal{N}'$. Par hypothèse, $\beta = \overrightarrow{cla}(x) = x_1 \vee x_2 \vee \cdots \vee x_k \vee x = \overline{exp(x)} \vee x$. D'où $exp(x) = \{\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k}\} \subseteq \mathcal{N}$ (par définition du graphe d'implication). Puisque $\mathcal{N} \subseteq \mathcal{N}'$, par transitivité nous avons $exp(x) \subseteq \mathcal{N}'$;*
2. – *$\{(x, \bot), (\overline{x}, \bot)\} \subseteq \mathcal{A}'$. Par construction ;*
   – *$\{(\overline{y_1}, \overline{x}), (\overline{y_2}, \overline{x}) \ldots, (\overline{y_k}, \overline{x})\} \subseteq \mathcal{A}'$. Idem ;*
   – *$\{(\overline{x_1}, x), (\overline{x_2}, x), \ldots, (\overline{x_k}, x)\} \subseteq \mathcal{A}'$. Nous savons par hypothèse que $exp(x) = \{\overline{x_1}, \overline{x_2} \ldots, \overline{x_k}\}$ et que $\{exp(x), x\} \subseteq \mathcal{I}'$. D'où $\{(\overline{x_1}, x), (\overline{x_2}, x), \ldots, (\overline{x_k}, x)\} \subseteq \mathcal{A}$ par définition du graphe d'implications. Puisque $\mathcal{A} \subseteq \mathcal{A}'$, nous obtenons par transitivité de l'inclusion le résultat escompté ;*
3. *$\forall x \in \mathcal{N}$, si $x \neq z$ et $\alpha = \bigwedge\{y \in \mathcal{N} \mid (y, x) \in \mathcal{A}\} \not\models \bot$ alors $\overline{\alpha} \vee x \in \Sigma$ et est unisatisfaite en x. D'après la proposition 6.4, $\forall y \in \mathcal{N}$ tel que $y \neq x$ et $\overrightarrow{cla}(y) \neq \bot$ nous avons $\overrightarrow{cla}(y)$ unisatisfaite par $\mathcal{I}$ en y. Par construction de $\mathcal{G}_\Sigma^{\mathcal{I}}$ et $\mathcal{G}_{\mathcal{I},\Sigma}^{x}$ la propriété précédente est vérifiée.*

Exemple 6.7. *Reprenons l'exemple 6.1 et les interprétations partielles dérivées obtenues dans les exemples 6.5 et 6.6. Nous pouvons alors étendre les graphes d'implications associés à ces deux interprétations en deux graphes conflits schématisés dans les figures 6.3(a) (ordre lexicographique inverse) et 6.3(b) (ordre lexicographique).*



(a) Ordre lexicographique inverse



(b) Ordre lexicographique

FIGURE 6.3 – Graphe conflit construit à l'aide de la propagation unitaire. Les nœuds losanges et les arcs en pointillés représentent respectivement les nœuds et les arcs ajoutées au graphe d'implication classique afin de l'étendre en un graphe conflit.

Comme nous pouvons le voir sur l'exemple précédent deux cas peuvent survenir lors de la construction du graphe d'implication lié à l'interprétation partielle dérivée. Dans le premier cas (figure 6.3(a)), l'interprétation partielle dérivée conduit à un conflit et une analyse de conflits classique peut être effectuée. Dans second le cas, c'est-à-dire lorsque l'interprétation partielle dérivée diverge sur au moins un littéral ℓ (figure 6.3(b)), une raison pour le littéral $\neg\ell$ est ajoutée artificiellement permettant ainsi d'effectuer de nouveau une analyse de conflits traditionnelle.

Il est important de souligner qu'il n'est pas toujours possible d'étendre un graphe d'implication issu d'une interprétation partielle dérivée en un graphe conflit. En effet, comme nous pouvons le voir dans l'exemple suivant, dans certaines configurations il est impossible de trouver une clause unisatisfaite par l'interprétation complète $\mathcal{I}$ pour le littéral conflit.

Exemple 6.8. *Considérons la formule* $\Sigma = \{(a \vee b), (\neg a \vee \neg b), (\neg c \vee a), (\neg c \vee b), (e \vee d)\}$, $\mathcal{I} = \{\neg a, \neg b, c, d, e\}$ *une interprétation complète conflictuelle et* $\delta = [c, a, b, d, e]$ *une séquence de décision. L'interprétation partielle dérivée obtenue à partir de* δ *est* $\mathcal{I}' = \{(e), a\}$ *et le littéral conflit est* a*. Comme nous pouvons le voir il n'existe aucune clause de* Σ *unisatisfaite par l'interprétation complète* $\mathcal{I}$ *en* a*.*

Cependant, comme il est énoncé dans la propriété suivante, il est toujours possible d'étendre un graphe d'implication obtenu à partir d'une interprétation partielle dérivée en un graphe conflit sur le littéral conflit lorsque l'interprétation complète utilisée pour construire l'interprétation partielle dérivée se trouve dans un minimum local.

Propriété 6.6. *Soient Σ une formule* CNF*, $\mathcal{I}$ une interprétation complète conflictuelle construite sur $\mathcal{V}_\Sigma$, $\mathcal{I}'$ une interprétation partielle dérivée telle que x est le littéral conflit. Si $\mathcal{I}$ se trouve dans un minimum local alors $\exists \alpha \in \Sigma$ unisatisfaite par $\mathcal{I}$ en x.*

Preuve 6.6. *Raisonnons par l'absurde et supposons que $\nexists \alpha \in \Sigma$ unisatisfaite en x par l'interprétation $\mathcal{I}$. Par définition de l'interprétation partielle dérivée $x \in \mathcal{I}$. Soit $\mathcal{I}_x$ l'interprétation voisine de $\mathcal{I}$ qui consiste à inverser la valeur de vérité de la variable x dans $\mathcal{I}$. Puisqu'il n'existe aucune clause de Σ unisatisfaite en x par l'interprétation $\mathcal{I}$, $\{\alpha \in \Sigma | \mathcal{I}_x \not\models \alpha\} \subseteq \{\alpha \in \Sigma | \mathcal{I} \not\models \alpha\}$. D'après la proposition 6.3 $\exists \beta \in \Sigma$ falsifiée par $\mathcal{I}$ telle que $\neg x \in \beta$ et par conséquent $\mathcal{I}_x \models \beta$. Donc $\beta \notin \{\alpha \in \Sigma | \mathcal{I}_x \not\models \alpha\}$ et par conséquent le nombre de clauses falsifiées par $\mathcal{I}$ est strictement supérieur au nombre de clauses falsifiées par $\mathcal{I}_x$. Absurde puisque $\mathcal{I}$ se trouve dans un minimum local.*

L'algorithme 6.3 illustre une analyse de conflits basée sur la notion d'interprétation partielle dérivée. En plus d'analyser et d'extraire un *nogood*, cette fonction ajuste les littéraux qui diffèrent entre l'interprétation complète et l'interprétation partielle dérivée. De cette manière, après l'appel de cette fonction, la formule obtenue par propagation unitaire des littéraux de l'ensemble conflit est cohérente (modulo la propagation unitaire).

Algorithme 6.3 : Analyse de conflit à partir d'une interprétation partielle dérivée

Données : Σ une formule et $\mathcal{I}$ une interprétation complète conflictuelle se trouvant dans un minimum local
Résultat : β une clause impliquée par Σ et l'interprétation $\mathcal{I}$ modifiée

1 **Début**
2 $\quad \alpha \longleftarrow \bot$; `/* la clause conflit */`
3 $\quad \beta \longleftarrow \bot$; `/* la clause retournée */`
4 $\quad \mathcal{I}' \longleftarrow \emptyset$; `/* Interprétation partielle dérivée */`
5 $\quad \mathcal{S} \longleftarrow \emptyset$; `/* la dernière séquence de décision propagation */`
6 $\quad$ **tant que** $(\mathcal{I}' \subseteq \mathcal{I})$ *et* $(\alpha = \bot)$ **faire**
7 $\quad\quad \ell \longleftarrow$`choixLittéral(`$\mathcal{I} \setminus \mathcal{I}'$`);` `/* littéral non affecté */`
8 $\quad\quad \mathcal{I}' \longleftarrow \mathcal{I}' \cup \{\ell\}$;
9 $\quad\quad \alpha \longleftarrow$`propagationUnitaire(`$\Sigma, \mathcal{I}', \mathcal{S}$`)` [a];
10 $\quad$ **si** $\alpha \neq \bot$ **alors**
11 $\quad\quad \beta \longleftarrow$`analyseConflit(`$\Sigma, \mathcal{I}', \alpha$`);` `/* clause assertive */`
12 $\quad$ **sinon**
$\quad\quad$ `/* littéral qui diffère entre les interprétations` $\mathcal{I}$ `et` $\mathcal{I}'$ `*/`
13 $\quad\quad x_i \in \mathcal{S} = \langle (x) x_1, x_2, \ldots, x_n \rangle$ tel que $\neg x_i \in \mathcal{I}$ et $\nexists j, 1 \leqslant j < i, \neg x_j \in \mathcal{I}$;
14 $\quad\quad \gamma \longleftarrow \gamma \in \Sigma$ telle que γ est unisatisfaite par $\mathcal{I}$ en x_i;
15 $\quad\quad \beta \longleftarrow$`analyseConflit(`$\Sigma, \mathcal{I}', \eta[x_i, \alpha, \gamma]$`);` `/* clause assertive */`
16 $\quad \mathcal{I} \longleftarrow \mathcal{I}' \cup \{\ell \in \mathcal{I}$ tel que $\neg \ell \notin \mathcal{I}'\}$; `/* modification de` $\mathcal{I}$ `*/`
17 $\quad$ **retourner** β;
18 **Fin**

a. La fonction `propagationUnitaire` est modifiée pour retourner la séquence de décision propagation.

Cet algorithme prend en entrée une formule Σ sous CNF ainsi qu'une interprétation complète $\mathcal{I}$ se trouvant dans un minimum local (nécessaire pour être sûr de pouvoir étendre le graphe d'implications en un graphe conflit). En sortie, elle modifie l'interprétation $\mathcal{I}$ (ligne 16) et retourne une clause impliquée par Σ (ligne 17). Après la partie initialisation (lignes 2–5), l'interprétation partielle dérivée est construite. Pour cela, tant qu'un conflit n'a pas été obtenu par propagation unitaire (ligne 9) et que l'interprétation partielle $\mathcal{I}'$ est incluse dans $\mathcal{I}$, un littéral ℓ est choisi afin d'étendre l'interprétation $\mathcal{I}'$ et la propagation unitaire est effectuée (ligne 7–9). Le littéral ℓ est sélectionné à l'aide de la fonction `choixLittéral` parmi l'ensemble des littéraux de $\mathcal{I}$ n'apparaissant pas dans $\mathcal{I}'$ (ligne 7). Cette fonction, pour notre étude expérimentale qui est reportée dans la section 6.2.2.2, utilise le même principe que VSIDS (voir 3.1.4.3) pour choisir la prochaine variable à affecter (pondération des variables lors de l'analyse de conflits). Une fois l'interprétation partielle dérivée calculée, deux cas sont à considérer : (i) soit la clause α est différente de $\bot$ (un conflit est obtenu par propagation unitaire) et une analyse de conflits classique basée sur la notion de UIP peut être effectuée (ligne 10–11) ; (ii) soit le graphe d'implications induit par $\mathcal{I}'$ est étendu en un graphe conflit et une analyse de conflits classique est aussi effectuée (ligne 12–15). Dans les deux cas, l'interprétation complète est modifiée ($\mathcal{I}' \subseteq \mathcal{I}$) et le *nogood* obtenu par analyse de conflits est retourné.

6.2 Ajout de clauses dans un solveur de type WSAT

Dans cette section, nous présentons une approche de type WSAT (voir section 2.1.3) où une méthode d'analyse de conflits a été ajoutée afin de sortir des minima locaux. Une fois cette dernière définie, nous étudions expérimentalement son comportement vis-à-vis des deux méthodes d'analyses de graphe conflit présentées précédemment.

6.2.1 CDLS : algorithme

Nous proposons d'intégrer les deux procédures d'analyse et de construction de graphe conflit définies dans la section précédente au sein d'un solveur de type WSAT. Cette méthode, nommée CDLS (*Conflict Driven for Local Search*), consiste à effectuer une analyse de conflit lorsqu'un minimum local est atteint par la recherche locale. Une fois cette dernière effectuée, la clause obtenue est ajoutée à la base de connaissance et le poids des littéraux de celle-ci est augmenté ($makecount$ et $breakcount$, voir 2.1.4) dans le but de sortir du minimum local. L'algorithme 6.4 donne un schéma de cette approche. Il prend en paramètre une CNF Σ ainsi que deux entiers $maxReparations$ et $maxEssais$ représentant respectivement le nombre de réparations et le nombre d'essais autorisée pour la recherche locale. Il peut retourner trois valeurs : `vrai`, `faux` ou `échec`, selon que le problème a été montré satisfaisable, insatisfaisable ou s'il est impossible de conclure. Cet algorithme reprend le schéma de base de l'algorithme WSAT (voir algorithme 2.3). La différence entre CDLS et WSAT se situe au niveau du critère d'échappement. Contrairement à la méthode de base deux situations sont à considérer en fonction du résultat de l'évaluation du prédicat `faireAnalyse`. Si ce dernier retourne `faux` un critère d'échappement classique est utilisé (recherche locale classique, lignes 11–14). Dans le cas où `faireAnalyse` est évalué à `vrai` la fonction `analyseGrapheConflit` est appelée afin d'analyser et d'apprendre une nouvelle clause α (suivant l'un des schémas d'apprentissage présenté précédemment). Si cette dernière est égal à la clause vide, le problème est montré insatisfaisable et `faux` est retourné (ligne 9). Sinon, la clause α est ajoutée à la base de clauses apprises (ligne 10). Comme pour les solveurs SAT modernes, la fonction `réductionClausesApprises` (permettant de réduire la base de clauses apprises) est appelée

lorsque `faireRéduction` retourne `vrai` (ligne 15–16).

Algorithme 6.4 : CDLS

Données : Σ une formule, deux entiers $maxReparations$ et $maxEssais$
Résultat : `vrai` si la formule Σ est satisfiable, `faux` si la formule est prouvé insatisfiable et `échec` s'il est impossible de conclure

```
1  Début
2    Δ ⟵ ∅;                                   /* l'ensemble de clauses apprises */
3    pour i de 0 à maxEssais faire
4      I ⟵ une interprétation complète générée aléatoirement;
5      pour j de 0 à maxReparations faire
6        si (eval(Σ ∪ Δ, I) = 0) alors retourner vrai ;
7        si faireAnalyse() et I se trouve dans un minimum local [a] alors
8          α ⟵analyseGrapheConflit(Σ ∪ Δ, I);
9          si (α = ⊥) alors retourner faux ;          /* problème incohérent */
10         Δ ⟵ Δ ∪ α;                                  /* apprentissage */
11       sinon
           /* recherche locale classique                                  */
12         α ⟵ une contrainte au hasard parmi les contraintes falsifiées par I;
13         si (∃I_c ∈ N_WSAT(I, α) telle que diff(Σ ∪ Δ, I, I_c) > 0) alors I ⟵ I_c;
14         sinon I ⟵ interprétation choisie suivant un critère d'échappement;
15     si faireRéduction() alors
16       réductionClausesApprises(Δ);
17   si (eval(F, I) = 0) alors retourner vrai ;
18   retourner échec ;
19 Fin
```

a. Nous considérons un minimum local pour GSAT

Dans la suite, nous présentons la fonction `réductionClausesApprises` ainsi que les prédicats `faireRéduction` et `faireAnalyse` laissés en suspens lors de cette description.

6.2.1.1 Nettoyage de la base de clauses apprises

Pour la stratégie de nettoyage de la base de clauses apprises nous avons choisi de conserver principalement les clauses utilisées par la recherche locale. Pour cela, un compteur initialisé à 1 est associé à chaque clause lors de leur création (ligne 8). Ensuite, lorsqu'une clause est choisie (ligne 11) son poids est incrémenté. Les clauses supprimées sont alors celles qui ont un poids égal à zéro. Après cette étape de suppression, le poids de toutes les clauses est divisé par deux. De cette manière, le poids des clauses fréquemment utilisées au début de la recherche et qui ne sont plus utilisées sont supprimées.

6.2.1.2 Fréquence de nettoyage de la base de clauses apprises

Nous avons choisi pour la fréquence de nettoyage de la base de clauses apprises d'employer le même principe que celui utilisé dans le cadre des solveurs SAT modernes. Plus précisément, le prédicat `faireRéduction` est évalué de la même manière que pour le solveur GLUCOSE (voir 3.1.5.2). La

base de clauses apprises est réduite lorsque le nombre de clauses ajoutées depuis le dernier nettoyage est supérieur à $4000 + 1000 \times x$ où x représente le nombre d'appels à la fonction de réduction.

6.2.1.3 Appel à la fonction d'analyse de conflits

Lorsqu'une nouvelle clause est ajoutée à la base le paysage de recherche exploré par la méthode de recherche locale est modifié. Afin de pouvoir profiter de cette nouvelle information, il peut être intéressant de permettre à la recherche locale d'effectuer un certain nombre de réparations avant de considérer qu'elle se trouve en situation d'échec. Pour cela, nous considérons deux variables entières $nbMinimum$ et $bornMinimum$ représentant respectivement le nombre de minima locaux rencontré depuis le dernière appel à la fonction `analyseGrapheConflit` et le nombre de minima locaux à partir duquel la recherche locale est supposée être en difficulté. Le prédicat `faireAnalyse` consiste alors à retourner `vrai` si $nbMinimum > bornMinimum$ et `faux` sinon.

6.2.2 Études expérimentales

Nous étudions les performances de notre solveur CDLS vis-à-vis des deux implémentations de la fonction `analyseGrapheConflit` présentées précédemment (chemin critique et interprétation partielle dérivée). Pour chacune des deux versions nous étudions différentes propositions en relation avec les prédicats `faireRéduction` et `faireAnalyse`. Les solveurs résultants sont notés $\text{CDLS}_{(analyse,b,nb)}$ où : $analyse$ est soit $dérivée$ (6.1.3) soit $critique$ (6.1.2) en fonction que l'analyse du graphe conflit implémentée est basée sur la notion d'interprétation partielle dérivée ou de chemin critique respectivement ; b est une variable booléenne stipulant si la version du solveur considérée effectue ou non le nettoyage de la base de clauses apprises ; nb est la valeur d'initialisation de la variable $bornMinimum$. Par exemple, $\text{CDLS}_{(critique,\texttt{vrai},1000)}$ signifie que la version de CDLS testée initialise $bornMinimum$ à 1000, utilise la notion de chemin critique pour effectuer l'analyse du graphe conflit et effectue le nettoyage de la base clauses apprises. En ce qui concerne le critère d'échappement (ligne 14 de l'algorithme), nous avons choisi d'utiliser le stratégie WALKSAT. Par la suite, nous notons WALKSAT les versions de CDLS qui n'utilisent pas l'analyse de conflit ($\text{CDLS}_{(analyse,b,\infty)}$) .

L'ensemble des résultats expérimentaux reportés dans ce chapitre a été obtenu sur un Quad-core Intel XEON X5550 avec 32Gb de mémoire. Le temps CPU a été limité à 900 secondes. Les instances utilisées sont issues de la compétitions SAT 2009 (Le Berre et Roussel 2009). Elles sont divisées en trois catégories : *crafted* (281 instances), application (292) et aléatoire (570). Toutes ces instances sont prétraitées par SATELITE (Eén et Biere 2005).

6.2.2.1 CDLS : analyse de conflits basée sur la notion de chemin critique

Nous reportons dans un premier temps les résultats obtenus par notre méthode utilisant la notion de chemin critique dans le cas où la base de clauses apprises n'est pas nettoyée. Nous présentons ensuite les résultats obtenus lorsque le nettoyage de la base de clauses apprises est effectué.

Sans réduction de la base de clauses apprises Le tableau 6.1 reporte les résultats obtenus par les différentes version de CDLS n'effectuant pas de nettoyage de la base de clauses apprises et utilisant la notion de chemin critique. Sur ce tableau, nous pouvons voir que les résultats obtenus par la meilleure version de CDLS sont globalement identiques à ceux obtenus par la méthode de recherche locale. Néanmoins, nous remarquons que notre approche permet de résoudre légèrement plus d'instances structurées et un peu

moins d'instances aléatoires. Nous pouvons aussi remarquer que la valeur de la variable $bornMinimum$ influe grandement sur les performances de la méthode.

	Application			*Crafted*			Aléatoire			Total		
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
WALKSAT	3	3	6	38	1	39	**76**	0	**76**	**117**	4	**121**
CDLS$_{(critique,\texttt{faux},0)}$	4	**4**	8	20	**2**	22	54	0	54	78	**6**	84
CDLS$_{(critique,\texttt{faux},100)}$	**7**	3	**10**	29	**2**	31	59	0	59	95	5	100
CDLS$_{(critique,\texttt{faux},1000)}$	5	3	8	37	**2**	39	67	0	67	109	5	114
CDLS$_{(critique,\texttt{faux},5000)}$	5	3	8	**39**	**2**	**41**	72	0	72	116	5	**121**

TABLE 6.1 – Résultat obtenus par CDLS sans réduction de la base de clauses apprises et avec une analyse de conflits basée sur la notion de chemin critique.

Remarquons tout de même que, bien que les résultats obtenus par cette version de CDCL ne sont pas meilleurs (en terme de nombre d'instances résolus) que la méthode de recherche locale classique, nous pouvons voir sur la figure 6.4 que notre méthode est plus rapide (plus de points sous la diagonale) que WALKSAT.

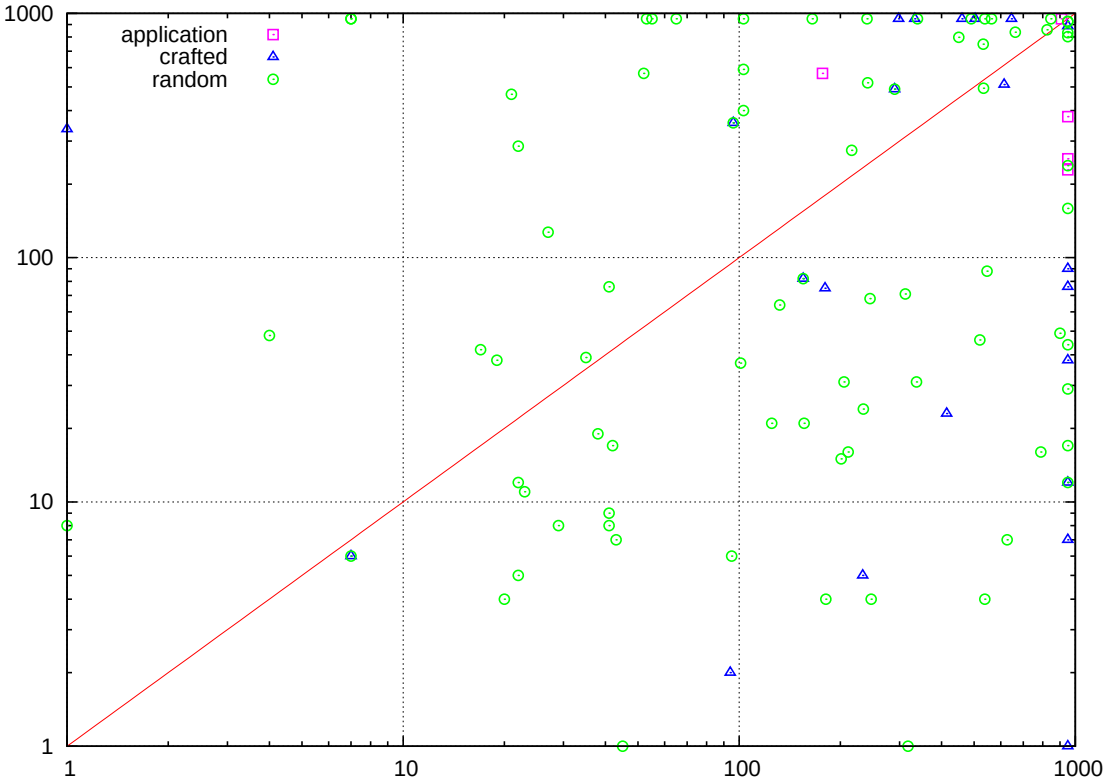


FIGURE 6.4 – Nuage de points : WALKSAT versus CDLS$_{(critique,\texttt{faux},5000)}$. Un point (temps WALKSAT, temps CDLS$_{(critique,\texttt{faux},5000)}$) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

Avec réduction de la base de clauses apprises Nous reportons dans le tableau 6.2 les résultats obtenus par différentes versions de CDLS n'effectuant pas de nettoyage de la base de clauses apprises et util-

isant la notion de chemin critique. Sur ce tableau nous pouvons voir que la meilleure version de CDLS (CDLS$_{(critique,\texttt{vrai},5000)}$) permet de résoudre globalement plus d'instances que la méthode de recherche locale classique. De plus, il est important de noter qu'une des versions permet d'obtenir les mêmes performances que la méthode de recherche locale classique sur les instances aléatoires.

	Application			*Crafted*			Aléatoire			Total		
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
WALKSAT	3	3	6	38	1	39	**76**	0	**76**	117	4	121
CDLS$_{(critique,\texttt{vrai},0)}$	**6**	**4**	**10**	20	**2**	22	64	0	64	90	**6**	96
CDLS$_{(critique,\texttt{vrai},100)}$	**6**	3	9	30	**2**	32	**76**	0	76	112	5	117
CDLS$_{(critique,\texttt{vrai},1000)}$	**6**	3	9	38	**2**	40	73	0	73	117	5	122
CDLS$_{(critique,\texttt{vrai},5000)}$	5	3	8	**44**	**2**	**46**	72	0	72	**121**	5	**126**

TABLE 6.2 – Résultat obtenus par CDLS avec réduction de la base de clauses apprises et avec une analyse de conflits basée sur la notion de chemin critique.

Comme pour la méthode n'effectuant pas de nettoyage de la base de clauses apprises, nous pouvons voir sur la figure 6.5 qu'ajouter notre technique d'analyse de conflits au sein de WALKSAT permet d'être plus rapide.

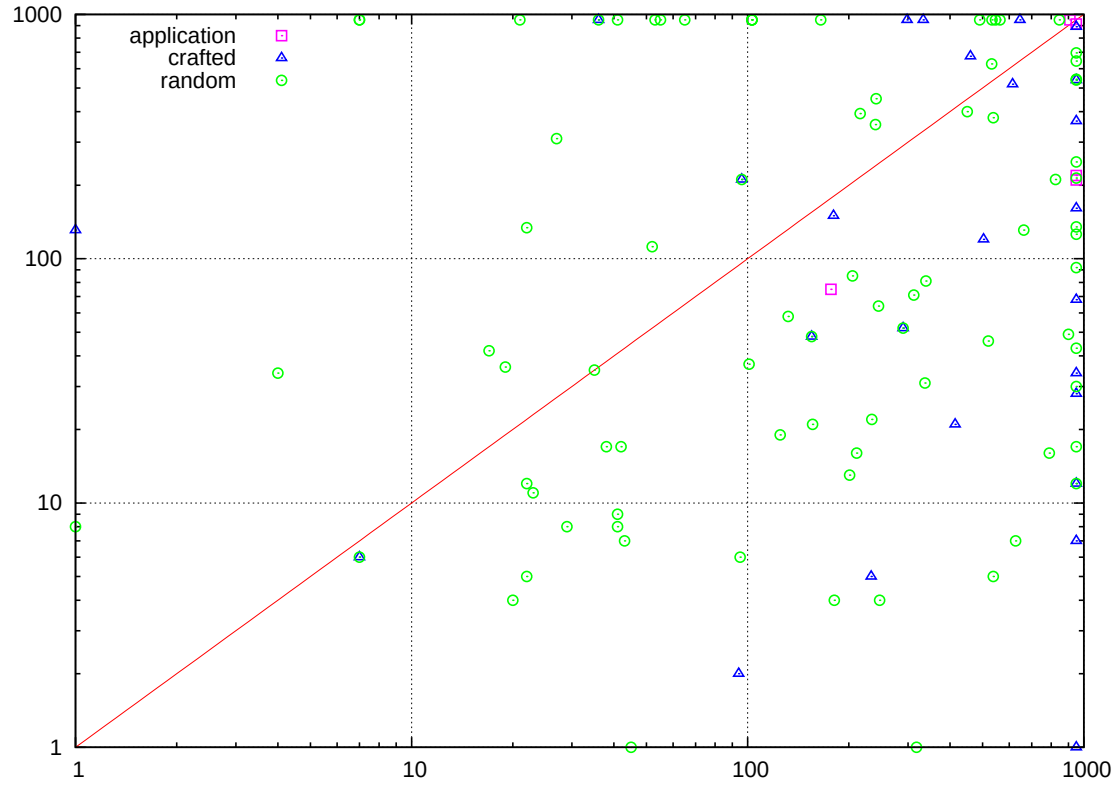


FIGURE 6.5 – Nuage de points : WALKSAT versus CDLS$_{(critique,\texttt{vrai},5000)}$. Un point (temps WALKSAT, temps CDLS$_{(critique,\texttt{vrai},5000)}$) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

Discussion L'étude expérimentale que nous venons de conduire a permis de mettre en évidence l'importance de la réduction de la base de clauses apprises et de la valeur d'initialisation de la variable $bornMinimum$. En effet, nous avons pu voir que pour chaque catégorie d'instances il existe une valeur optimale de la variable $bornMinimum$. En ce qui concerne le nettoyage de la base de *nogoods*, nous avons pu voir au travers des deux tableaux (6.1 et 6.2) que l'effectuer permettait d'être plus efficace. Ce résultat n'est pas surprenant puisque la taille de la formule influe énormément sur les performances de la méthode de recherche locale (l'inversion de la valeur de vérité d'une variable est plus coûteuse) et par conséquent sur les performances de notre solveur.

Un autre point important de cette étude concerne l'apport de l'analyse de conflits basée sur la notion de chemin critique. Nous avons pu voir lors de ces expérimentations que, bien que notre méthode permettait de résoudre plus rapidement, le gain obtenu en terme de nombre d'instances résolues était marginal (5 instances résolus en plus). Ce résultat peut en partie être expliqué par le fait que la méthode d'extraction de *nogood* présentée passe beaucoup de temps au calcul d'un chemin critique complet et donc n'effectue que très peu de réparations.

6.2.2.2 CDLS : analyse de conflits basée sur la notion d'interprétation partielle dérivée

Dans cette partie, nous étudions les performances obtenues par l'approche CDLS utilisant la notion d'interprétation partielle dérivée. Cette étude est menée de manière à considérer le cas où la base de clauses apprises est nettoyée et le cas où elle ne l'est pas.

Sans réduction de la base de clauses apprises Le tableau 6.3 reporte les résultats obtenus par différentes version de CDLS n'effectuant pas de nettoyage de la base de clauses apprises et utilisant la notion d'interprétation partielle dérivée afin d'analyser le graphe conflit. Sur ce tableau, nous pouvons voir que les résultats obtenus quelle que soit la version de CDLS considérée sont globalement supérieurs à ceux obtenus par la méthode de recherche locale. De plus, nous pouvons voir que notre méthode permet d'augmenter systématiquement le nombre d'instances résolues et cela quelle que soit la catégorie d'instances considérée. En effet, nous pouvons voir que, contrairement aux versions de CDLS basées sur la notion de chemin critique, l'analyse du graphe conflit basée sur la notion d'interprétation partielle dérivée permet d'accroître les performances de CDLS sur les instances aléatoires (2.5 fois plus d'instances aléatoires résolues). Concernant la valeur d'initialisation de la variable $bornMinimum$, nous pouvons voir sur ce tableau qu'elle influe de manière encore plus manifeste sur les performances de la méthode que lors des deux dernières expérimentations (voir 6.1 et 6.2). Nous pouvons effectivement voir que, pour certaines catégories de problèmes, le nombre d'instances résolues peut être jusque deux fois plus important en fonction de la valeur d'initialisation de la variable $bornMinimum$ choisie (application : $23 \rightarrow 57$, *crafted* : $41 \rightarrow 72$ et aléatoire : $137 \rightarrow 192$).

	Application			*Crafted*			Aléatoire			Total		
	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
WALKSAT	3	3	6	38	1	39	76	0	76	117	4	121
CDLS$_{(dérivée,\texttt{faux},0)}$	**27**	**30**	**57**	31	**10**	41	137	0	137	195	**40**	235
CDLS$_{(dérivée,\texttt{faux},100)}$	19	19	38	52	9	61	180	0	180	251	28	279
CDLS$_{(dérivée,\texttt{faux},1000)}$	17	10	27	57	8	65	**192**	0	**192**	**266**	18	**284**
CDLS$_{(dérivée,\texttt{faux},5000)}$	16	7	23	**65**	7	**72**	184	0	184	265	14	279

TABLE 6.3 – Résultat obtenus par CDLS sans réduction de la base de clauses apprises et avec une analyse de conflit basée sur la notion d'interprétation partielle dérivée.

Nous reportons dans la figure 6.6 une comparaison entre la meilleure version de CDLS et WALKSAT. Les résultats reportés sont sans équivoque puisque, en plus de résoudre davantage d'instances, notre méthode permet dans 90% des cas de déterminer plus rapidement la satisfaisabilité d'une formule que la méthode de recherche locale classique.

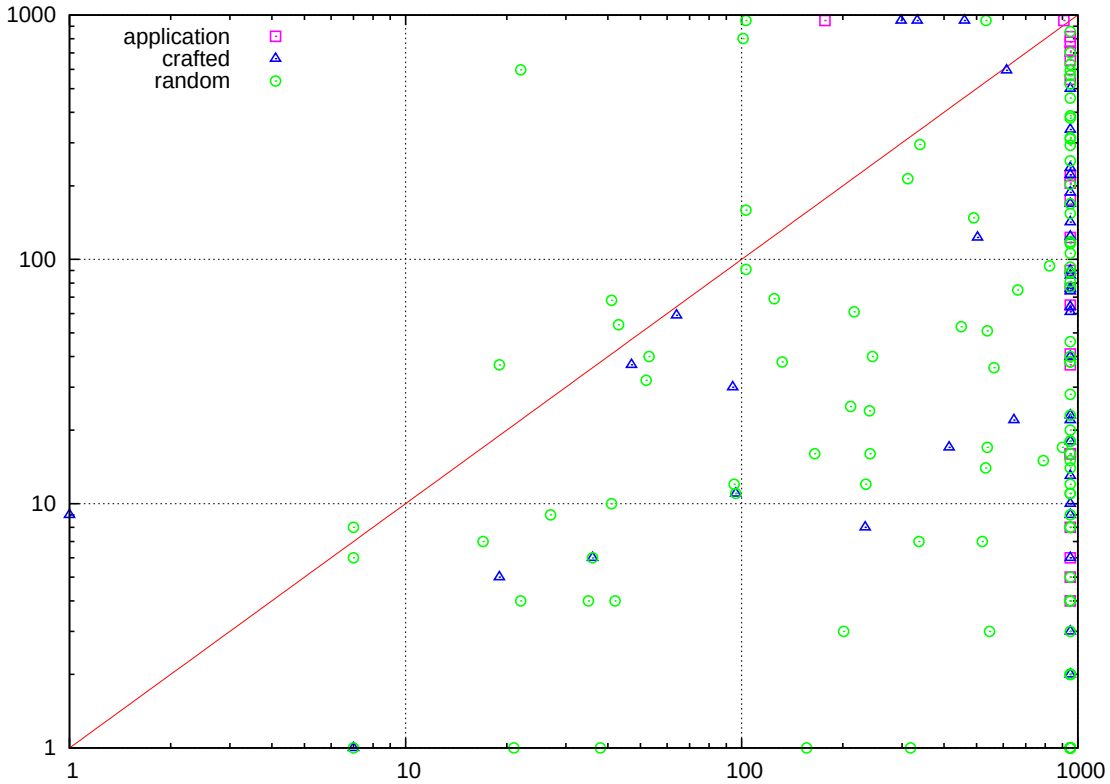


FIGURE 6.6 – Nuage de points : WALKSAT versus $\text{CDLS}_{(dérivée,\texttt{faux},1000)}$. Un point (temps WALKSAT, temps $\text{CDLS}_{(dérivée,\texttt{faux},1000)}$) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

Avec réduction de la base de clauses apprises Nous reportons dans le tableau 6.4 les résultats obtenus par différentes versions de CDLS utilisant la notion d'interprétation partielle dérivé et où la base de clauses apprises est nettoyée. Sur ce tableau, nous pouvons voir que cette version de CDLS obtient sensiblement les mêmes performances que la version étudiée précédemment et qui n'effectue pas de réduction de la base de *nogoods*. Cependant, il est important de noter que, comme pour la version de CDLS basée sur la notion de chemin critique, le nettoyage de la base de clauses apprises permet d'améliorer les performances de notre méthode.

	Application			*Crafted*			Aléatoire			Total		
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
WALKSAT	3	3	6	38	1	39	76	0	76	117	4	121
$\text{CDLS}_{(dérivée,\texttt{vrai},0)}$	**27**	**35**	**62**	36	**10**	46	157	0	157	220	**45**	265
$\text{CDLS}_{(dérivée,\texttt{vrai},100)}$	19	23	42	57	9	66	**195**	0	**195**	**271**	32	**303**
$\text{CDLS}_{(dérivée,\texttt{vrai},1000)}$	21	11	32	59	9	68	188	0	188	268	20	288
$\text{CDLS}_{(dérivée,\texttt{vrai},5000)}$	17	8	25	**64**	7	**71**	188	0	188	269	15	284

TABLE 6.4 – Résultat obtenus par CDLS avec réduction de la base de clauses apprises et avec une analyse de conflits basée sur la notion d'interprétation partielle dérivée.

Pour terminer, nous reportons dans la figure 6.7 l'étude comparative entre WALKSAT et la meilleur version de CDLS. Sur cette figure, nous pouvons voir que, comme pour les autres versions de CDLS étudiées, notre approche permet de résoudre les instances plus rapidement.

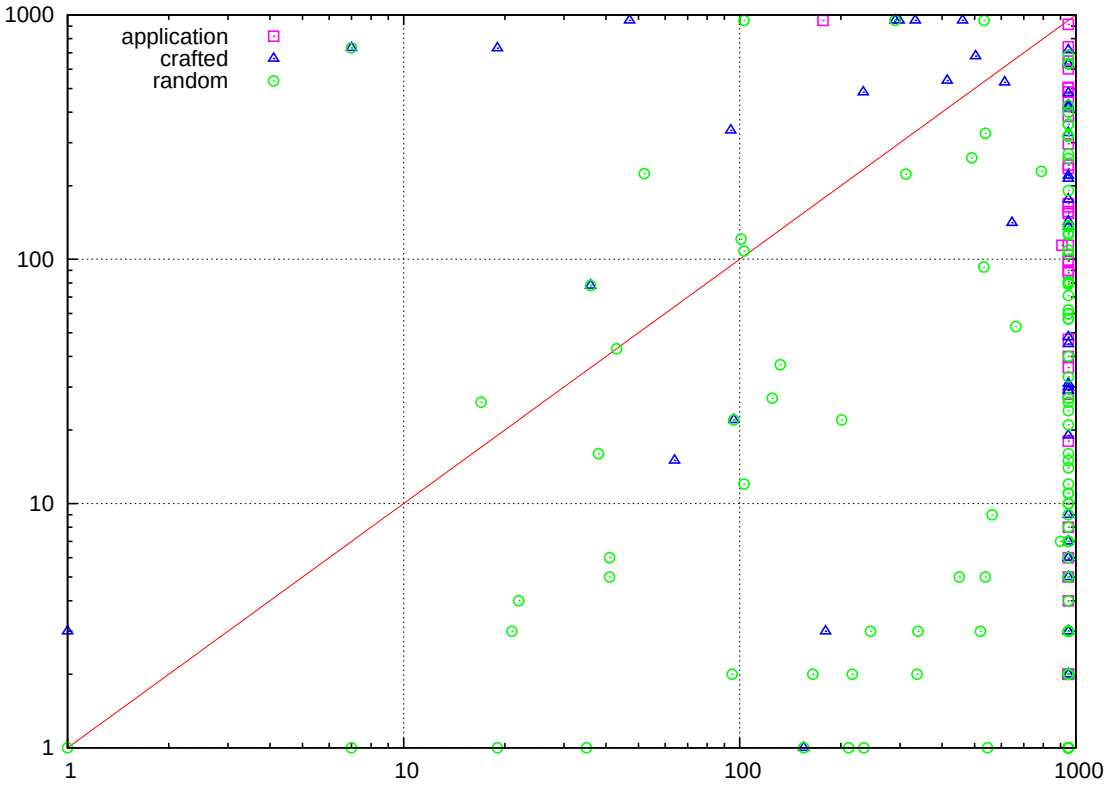


FIGURE 6.7 – Nuage de points : WALKSAT versus CDLS$_{(dérivée,\texttt{vrai},100)}$. Un point (temps WALKSAT, temps CDLS$_{(dérivée,\texttt{vrai},100)}$) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

Discussion Nous avons pu voir lors de ces deux dernières expérimentations que la version de CDLS utilisant la notion d'interprétation partielle dérivée obtient systématiquement de meilleures performances que la méthode de recherche locale WALKSAT. De plus, nous pouvons aussi remarquer que, comme pour la version de CDLS basée sur la notion de chemin critique, le nettoyage de la base de clauses apprises ainsi que la valeur d'initialisation de la variable $bornMinimum$ ont une grande influence sur les performances globales de la méthode. Ceci peut être expliqué par le fait que, quelle que soit la méthode d'analyse du graphe conflit considérée, les arguments avancés dans la discussion précédente sont encore valables. Pour terminer, il est important d'insister sur le fait que cette version de CDLS permet de monter l'insatisfaisabilité de près de 45 instances (dont 35 de la catégorie application).

6.2.2.3 Synthèse

Lors de cette première série d'expérimentations, nous avons pu voir que, quelle que soit la méthode d'analyse du graphe conflit considérée, le solveur CDLS permettait de résoudre plus efficacement les

problèmes structurés. Ces expérimentations ont aussi permis de mettre en évidence l'impact de la méthode de réduction de la base de *nogoods* et de la valeur d'initialisation de la variable $bornMinimum$ sur les performances générales de l'algorithme CDLS.

En ce qui concerne la version de CDLS utilisant la notion de chemin critique, bien que le bilan en nombre d'instances résolues soit mitigé, les premiers résultats obtenus sont très encourageant. En effet, malgré le fait que le temps nécessaire pour l'extraction d'un *nogood* soit important, cette version a montré qu'elle permettait de résoudre plus rapidement les instances et cela quelle que soit la catégorie.

Finalement, ces expérimentations ont aussi permis de mettre en exergue le fait que la version de CDLS basée sur la notion d'interprétation partielle dérivée obtient de bon résultats. Afin de vérifier si cette approche est compétitive, nous effectuons dans la section suivante une étude comparative entre la meilleure version de CDLS (CDLS$_{(dérivée,\texttt{vrai},100)}$) et un ensemble de solveurs de l'état de l'art. Dans la suite de ce chapitre et afin d'alléger les notations, nous notons simplement CDLS la méthode représentée par CDLS$_{(dérivée,\texttt{vrai},100)}$.

6.3 Étude comparative

Dans cette section, nous comparons CDLS avec certains solveurs de l'état de l'art. Plus précisément, notre approche a été comparée avec :

– deux méthodes de recherche locale :
  1. ADAPTG^{2}WSAT (Li *et al.* 2007) ;
  2. GNOVELTY+ (Pham et Gretton 2009) ;
– trois approches hybrides :
  1. HYBRIDGM (Balint *et al.* 2009) ;
  2. HINOTOS (Letombe et Marques-Silva 2008) ;
  3. CLS (Fang et Ruml 2004) ;
– une approche complète : MINISAT (Sörensson et Eén 2009).

Le tableau 6.5 résume, sur les trois catégories d'instances de la compétition SAT 2009, les résultats de CDLS et des solveurs énumérés précédemment. Rappelons, tout d'abord que ADAPTG^{2}WSAT et GNOVELTY+ sont des méthodes qui ne peuvent répondre qu'à la satisfaisabilité d'une instance. Notons aussi que la seule méthode similaire à la notre est CLS (voir 4.3.1.1). En effet, les autres approches sont soit des méthodes basées sur une méthode complète (HINOTOS, MINISAT et HYBRIDGM), soit des méthodes de recherche locale classique.

	Application			*Crafted*			Aléatoire			
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	Total
ADAPTG^{2}WSAT	8	3	11	68	1	69	**294**	0	**294**	**374**
GNOVELTY+	7	3	10	54	1	55	281	0	281	346
CDLS	19	23	42	57	9	66	195	0	195	303
CLS	5	4	9	28	3	31	140	0	140	180
HYBRIDGM	5	3	5	51	5	56	**294**	0	**294**	353
HINOTOS	39	68	107	69	**36**	**105**	65	**11**	77	288
MINISAT	**68**	**106**	**147**	**72**	27	99	3	0	3	249

TABLE 6.5 – Comparaison du solveur CDLS vis-à-vis d'un ensemble de solveurs de l'état de l'art. Pour chaque catégorie de problème et pour chaque solveur, nous reportons le nombre d'instances satisfaisables, insatisfaisables et total résolues.

En ce qui concerne la catégorie *crafted* l'analyse du tableau 6.5 indique que, les solveurs MINISAT et HINOTOS (lequel est basé sur MINISAT 1.4) mis à part, notre approche est compétitive et résout sensiblement le même nombre d'instances que les approches de recherche locale récentes. Nous pouvons aussi voir sur ce dernier que notre méthode est sensiblement meilleure que CLS sur ce type d'instances (deux fois plus d'instances résolues).

Pour les instances de la catégorie application, CDLS résout deux fois plus d'instances SAT que les solveurs stochastiques classiques et quatre fois plus que la méthode CLS. De plus, notre méthode permet aussi de résoudre plus d'instances insatisfaisables que cette dernière. Ces résultats montrent que l'analyse des conflits dans le cadre de la recherche locale, telle que nous l'avons défini précédemment, permet de résoudre des instances structurées, qu'elles soient satisfaisables ou pas.

Au niveau des instances aléatoires satisfaisables, CDLS est moins efficace que les solveurs stochastiques et HYBRIDGM (solveur hybride basée sur MARCH$_{ks}$ et GNOVELTY+, et qui est dédié à la résolution d'instances aléatoires). Ceci peut s'expliquer d'une part par le fait que la construction du graphe conflit est coûteuse en temps et qu'elle ralentit notre méthode. De ce fait, CDLS parcourt moins d'interprétations que les autres méthodes de recherche locale et a donc moins de chance de trouver un modèle. Et, d'autre part, par le fait que notre de méthode de recherche locale (WALKSAT) a des performances largement en deçà des meilleurs solveurs de recherche locale. Néanmoins, malgré cela nous pouvons voir que notre solveur est plus compétitif que MINISAT, HINOTOS et surtout CLS.

Pour terminer cette étude comparative, nous reportons dans la figure 6.8 le nuage de points permettant de comparer, en terme de temps de résolution, les solveurs CLS et CDLS. Sur cette figure nous pouvons clairement voir que notre méthode résout sensiblement plus d'instance que CLS et surtout plus rapidement (beaucoup plus de points se trouvent en dessous de la diagonale).

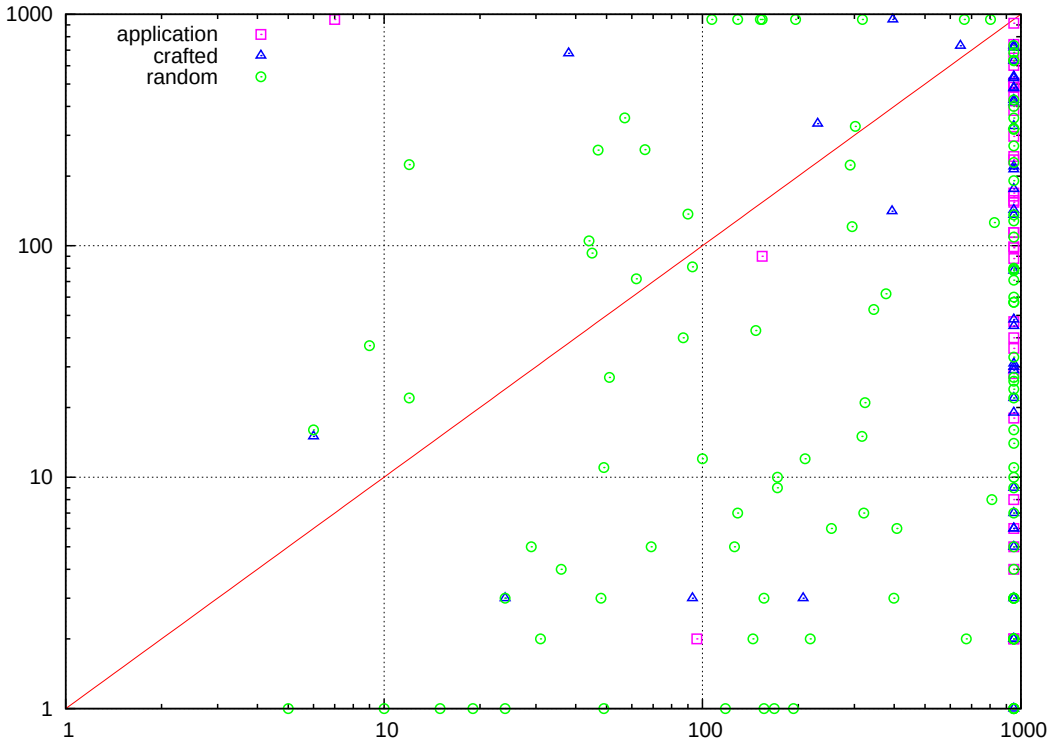


FIGURE 6.8 – Nuage de points : CLS versus CDLS. Un point (temps CLS, temps CDLS) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

6.4 Conclusion

Dans ce chapitre nous avons proposé une nouvelle approche pour sortir des minima locaux dans le cadre des méthodes de recherche locale pour SAT. Notre approche est basée sur une extension de l'analyse des conflits utilisée par les solveurs SAT modernes complets. L'objectif est double : améliorer les méthodes de recherche locale sur les problèmes structurés et répondre à un des challenges les plus importants de la communauté SAT, à savoir, proposer une méthode incomplète efficace répondant à l'insatisfaisabilité d'une formule. Les premiers résultats obtenus sur un large panel d'instances sont très encourageants. En effet, notre solveur CDLS est capable de résoudre des instances insatisfaisables, de plus, il est beaucoup plus efficace que les autres méthodes de recherche et même que certains hybrides pour résoudre des problèmes de la catégorie application. Néanmoins, CDLS n'obtient pas encore les performances des meilleures approches CDCL. Cette différence de performance mesure tout l'effort qu'il reste à fournir sur le développement de cette méthode. Les pistes que nous envisageons d'explorer dans l'avenir sont multiples.

La première piste que nous souhaitons explorer concerne la valeur d'initialisation de la variable $bornMinimum$. En effet, nous avons pu voir lors la présentation des résultats expérimentaux que la valeur optimale de cette dernière varie fortement par rapport à la catégorie d'instances considérée. Par la suite, nous comptons proposer une méthode permettant d'initialiser cette valeur dynamiquement en fonction du type d'instances à résoudre. Pour cela, nous envisageons dans un premier temps d'étudier le mécanisme de sélection de méthodes implémenté dans le solveur SATZILLA (Xu *et al.* 2008).

Lors de l'étude comparative nous avons pu estimer à quel point les performances de notre méthode de recherche locale était en deçà des performances obtenues par les meilleures méthodes de recherche locale. Cependant, nous avons pu constater que malgré cela notre solveur CDLS obtient des résultats très honorables. Par conséquent, il serait intéressant de greffer notre analyse de conflits à d'autres types d'algorithme de recherche locale, comme ADAPTG^{2}WSAT, afin d'améliorer leur performance.

En ce qui concerne l'exploitation de la notion de chemin critique plusieurs pistes peuvent être explorées. Par exemple il pourrait être intéressant de réduire la taille de la séquence de clauses retournées par la fonction d'extraction de chemin critique (algorithme 6.1) et par conséquent le nombre de résolutions effectuées par la méthode d'extraction de *nogood* (algorithme 6.2). Nous envisageons aussi de proposer une heuristique afin de sélectionner les clauses présentes dans le chemin critique (à l'heure actuelle ce choix est effectué de manière aléatoire). Pour terminer, nous étudions aussi la possibilité d'apprendre un ensemble de clauses plutôt qu'une seule. Cet ensemble pourrait par exemple être obtenu en considérant toutes ou certaines clauses obtenues par résolution lors du processus d'extraction de *nogoods*.

Une autre piste sur laquelle nous envisageons de travailler est liée à la stratégie de nettoyage de la base de clauses apprises utilisée dans le cadre du solveur CDLS. En effet, à la vue des différences de performances obtenues par les différentes version de CDLS utilisant ou pas la stratégie de nettoyage de la base de clauses apprises, il n'est pas envisageable de se passer de cette fonction. Cependant, il est très difficile de savoir quelle clause sera utile à l'avenir et par conséquent quelles sont les clauses à conserver. La méthode que nous proposons dans ce chapitre est basée sur le principe que si une clause n'est pas falsifiée lors du processus de recherche locale alors elle n'a pas de raison d'être conservée. Comme nous avons pu le voir, cette méthode obtient de bons résultats puisqu'elle permet d'améliorer systématiquement les performances de CDLS. Néanmoins, cette dernière ne permet pas de prendre en compte le fait que le problème est de grande taille (phase de descente très longue) ou le fait que la méthode de recherche locale reste engluée dans certains minima (supprimer les clauses n'appartenant pas à ces minima peut accentuer le phénomène).

Pour terminer, notons que bien que capable de montrer qu'une instance n'admet pas de modèle, l'approche CDLS basée sur la notion d'interprétation partielle dérivée reste incomplète. Dans le chapitre suivant, nous étendons la notion d'interprétation partielle dérivée et proposons un modèle algorithmique hybride coopératif basée sur cette dernière.

Chapitre 7

Solveur hybride pour la résolution pratique de SAT

Sommaire

BIEN QUE l'approche CDLS, introduite dans le chapitre précédent, soit capable de montrer qu'une instance n'admet pas de modèle, elle reste incomplète. Dans ce chapitre, nous proposons une nouvelle approche hybride, nommée SATHYS (*SAT Hybrid Solver*), pour le problème de la satisfiabilité propositionnelle. Cette approche combine la recherche locale et un solveur CDCL. Le solveur de recherche locale est le cœur de SATHYS et appelle le solveur CDCL pour fixer l'affectation d'un ensemble de variables lorsqu'un minimum local est atteint. Les deux moteurs collaborent étroitement et échangent des informations utiles l'un pour l'autre. Pour des instances satisfiables le fait de fixer des variables revient à les considérer comme tabou. Dans le cas où l'instance est insatisfiable, les informations obtenues par la recherche locale permettent au solveur CDCL de focaliser sa recherche sur un sous-ensemble de formules insatisfiables.

Ce chapitre est organisé de la manière suivante. Après avoir identifiées les informations à partager entre les deux méthodes, nous décrivons de manière formelle notre solveur hybride SATHYS. Avant de conclure et de donner quelques perspectives, nous étudions expérimentalement l'influence de la méthode de recherche locale et l'apport de l'échange d'informations dans notre solveur. Nous terminons cette étude en comparant notre solveur et d'autre solveurs de l'état de l'art.

Ces travaux ont fait l'objet de plusieurs publications (Audemard *et al.* 2009b; 2010a;c;b).

7.1 Échange bidirectionnel d'informations entre la recherche locale et une approche de type CDCL

Comme nous l'avons souligné dans le chapitre 4, la majorité des approches hybrides proposées dans la littérature s'appuient sur un schéma intégratif de combinaisons (voir section 4.3). Ces approches

utilisent pour la plupart une des méthodes comme esclave, c'est-à-dire que les informations échangées ne circulent que dans un sens (vers le maître). Bien que notre méthode soit aussi une approche intégrative (le cœur de notre solveur est une recherche locale), nous proposons un mécanisme où chaque méthode apporte des informations utiles à l'autre. Pour cela, nous commençons par identifier les informations qui pourrait être intéressantes de faire transiter.

7.1.1 Apport de la recherche locale pour une méthode de type CDCL

Comme nous avons pu le voir lors de la description des algorithmes complets, les solveurs CDCL sont composés de plusieurs éléments (heuristique de choix de variables, heuristique de choix de polarité, apprentissage, propagation unitaire, *etc.*.). Ces derniers sont très importants puisqu'ils conditionnent les performances globales du solveur. Dans cette partie, nous proposons deux approches basées sur la recherche locale afin de guider un solveur CDCL. La première approche consiste à utiliser la recherche locale afin d'ajuster dynamiquement l'heuristique de choix de variables. Pour cela nous utilisons la notion de variable frontière introduite par Goldberg (2009) :

Définition 7.1 (point frontière (Goldberg 2009))**.** *Soient Σ une formule* CNF *et $\mathcal{I}$ une interprétation complète (point) construite sur les variables de Σ. L'interprétation $\mathcal{I}$ est un point frontière si et seulement si $\exists \ell \in \mathcal{L}_\Sigma$ tel que $\forall \alpha \in \Sigma$ avec $\mathcal{I} \not\models \alpha$ nous avons $\ell \in \alpha$. Le littéral ℓ est appelé littéral frontière.*

Exemple 7.1. *Considérons la formule* CNF $\Sigma = \{\phi_1, \phi_2, \ldots, \phi_{11}\}$ *telle que :*

$$\begin{array}{llll} \phi_1 : (\neg x_1 \vee \neg x_2 \vee \neg x_3) & \phi_2 : (x_1 \vee \neg x_4 \vee \neg x_5) & \phi_3 : (x_2 \vee \neg x_1) & \phi_4 : (x_4 \vee \neg x_7 \vee \neg x_6) \\ \phi_5 : (x_3 \vee \neg x_5) & \phi_6 : (x_5 \vee \neg x_7) & \phi_7 : (x_6 \vee \neg x_8) & \phi_8 : (x_7 \vee \neg x_8) \\ \phi_9 : (x_8 \vee \neg x_4) & \phi_{10} : (x_1 \vee \neg x_8) & \phi_{11} : (x_7 \vee \neg x_9) & \phi_{12} : (\neg x_1 \vee \neg x_7 \vee \neg x_9) \end{array}$$

Soit l'interprétation complète $\mathcal{I}$ construite sur $\mathcal{V}_\Sigma$ telle que $\mathcal{I}(x_i) = \top$ pour tout $x_i \in \mathcal{V}_\Sigma$. L'ensemble des clauses falsifiées par $\mathcal{I}$ est $\{(\neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg x_1 \vee \neg x_7 \vee \neg x_9)\}$. L'interprétation $\mathcal{I}$ est donc un point frontière de Σ et le littéral frontière est $\neg x_1$.

Remarque 7.1 (variable frontière)**.** *Goldberg (2009) montre que pour une interprétation complète $\mathcal{I}$ qui est un point frontière d'une formule Σ, si $\ell \in \mathcal{L}_\Sigma$ est un littéral frontière de $\mathcal{I}$ pour Σ alors l'interprétation $\mathcal{I}_\ell$ qui consiste à inverser la valeur de vérité de ℓ dans l'interprétation $\mathcal{I}$ est soit un modèle de Σ, soit un point frontière de Σ tel que $\neg\ell$ est un littéral frontière. Dans le cas où ℓ et $\neg\ell$ sont des littéraux frontières nous parlons de variable frontière.*

Exemple 7.2. *Considérons l'exemple précédent, l'interprétation $\mathcal{I}_{x_1}$ qui consiste à inverser la valeur de vérité de x_1 dans $\mathcal{I}$ falsifie les clauses $(x_1 \vee \neg x_8)$ et $(x_1 \vee \neg x_4 \vee \neg x_5)$. L'interprétation $\mathcal{I}_{x_1}$ est donc un point frontière de Σ et par conséquent x_1 est une variable frontière.*

Dans (Goldberg 2009), l'auteur montre qu'appliquer une méthode de résolution classique tout en focalisant les opérations de résolution sur les variables frontières permet d'améliorer sensiblement la qualité de la preuve par résolution, c'est-à-dire de réduire la taille de l'arbre de réfutation. Ce résultat n'est pas surprenant puisque, comme l'énonce la propriété suivante, les variables frontières sont étroitement liées à la notion de noyau minimalement inconsistant.

Propriété 7.1. *Soient Σ une formule* CNF *inconsistante composée de $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ noyaux minimalement inconsistant et x une variable frontière de Σ, alors $\forall i, 1 \leqslant i \leqslant n, \exists \gamma \in \Gamma_i$ telle que $x \in \gamma$.*

Preuve 7.1. *D'après la propriété 1.4, si un problème est inconsistant alors quelque soit l'interprétation complète considérée au moins une clause de chaque* MUS *est falsifiée par cette interprétation. Soit $\mathcal{I}$ un point frontière de Σ en x. Puisque l'interprétation $\mathcal{I}$ est un point frontière de Σ, la variable x appartient à toutes les clauses de Σ falsifiées par $\mathcal{I}$ et par conséquent à au moins une clause de chaque* MUS *de Σ.*

Puisqu'elles apparaissent dans tous les MUS, les variables frontières peuvent être considérées comme très intéressantes pour conduire à l'établissement de la preuve de l'incohérence d'une formule. Cependant établir si une variable est une variable frontière est calculatoirement difficile. En effet, le problème qui consiste à vérifier si une contrainte donnée appartient à au moins un MUS est Σ_2^p (Eiter et Gottlob 1992). De plus, une formule peut posséder un nombre exponentiel de MUS. Ce qui implique que décider si une variable est frontière en calculant l'ensemble des MUS du problème est irréalisable dans le pire des cas. Pour pallier ce problème, nous proposons d'utiliser les interprétations complètes parcourues par la recherche locale afin de détecter des variables frontières. En effet, puisque la recherche locale tente autant que faire se peut de réduire le nombre de clauses falsifiées, il existe une forte probabilité que certaines des interprétations qu'elle parcourt soient des points frontières.

Le second point sur lequel la recherche locale peut venir en aide à une approche CDCL concerne l'heuristique de choix de polarité. En effet, les méthodes de recherche locale ont, contrairement aux solveurs CDCL, une vision globale de la formule. Elles peuvent donc « facilement » découvrir une interprétation complète de bonne qualité, c'est-à-dire falsifiant peu de contraintes du problème. Une telle interprétation peut ensuite être utilisée au sein d'une méthode de type CDCL afin de guider l'heuristique de choix de polarité. Cette méthode peut être vue comme un raffinement de l'heuristique de choix de polarité *progress saving* (voir 3.1.4.4) grâce à la recherche locale.

La figure 7.1 récapitule les informations que la méthode de recherche locale transmet au solveur CDCL.

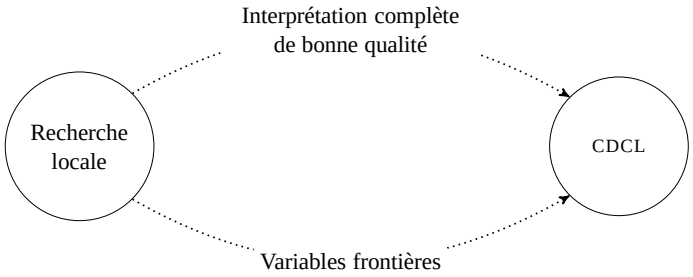


FIGURE 7.1 – Informations collectées par une méthode de recherche locale afin de guider un solveur de type CDCL.

7.1.2 Apport d'une méthode de type CDCL pour la recherche locale

Comme nous avons pu le voir lors de la présentation des algorithmes hybrides (voir chapitre 4), les algorithmes complets peuvent venir en aide à la recherche locale de différentes manières. Dans le cadre du solveur SATHYS nous avons exploré et étudié expérimentalement trois pistes :

– la première consiste à utiliser l'interprétation partielle générée par l'approche CDCL afin de limiter l'espace de recherche exploré par la méthode de recherche locale. L'avantage est double : premièrement, les variables fixées par l'interprétation partielle peuvent faire office de liste tabou. De cette manière la méthode de recherche locale considérée devient une méthode tabou où la liste

des variables interdites est gérée de manière dynamique grâce à la propagation unitaire et les retours arrières non chronologique. Par conséquent, la méthode complète peut être vue comme un critère d'échappement pour la recherche locale. Deuxièmement, la propagation unitaire permet de capturer un certain nombre de dépendances fonctionnelles. Récupérer de telles informations permet, comme cela a été démontré dans (Selman *et al.* 1997, Pham *et al.* 2007b, Kroc *et al.* 2009), d'améliorer sensiblement les méthodes de recherche locale sur les problèmes comportant une certaine structure ;

– la seconde piste consiste à utiliser la méthode complète pour ajouter des clauses à la formule initiale afin, d'une part de pouvoir résoudre les problèmes inconsistants, et d'autre part de guider la recherche locale en redéfinissant la fonction d'évaluation. Cette approche a en quelque sorte déjà été validée expérimentalement dans le chapitre précédent avec CDLS, puisque comme nous avons pu le voir l'apprentissage permet d'améliorer sensiblement les performances des solveurs de recherche locale sur les problèmes structurés ;
– la dernière piste consiste à utiliser la propagation unitaire afin d'initialiser l'interprétation complète utilisée par le solveur de recherche locale.

La figure 7.2 récapitule les informations que le solveur CDCL transmet à la recherche locale.

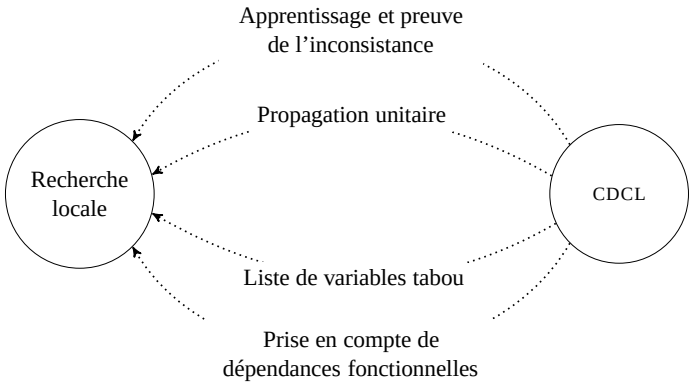


FIGURE 7.2 – Informations collectées par un solveur CDCL afin de guider une méthode de recherche locale.

7.2 SATHYS, un nouvel algorithme hybride pour la résolution pratique de SAT

Dans cette section nous décrivons de manière formelle notre solveur hybride SATHYS. Nous présentons ensuite les différentes méthodes et mécanismes mis en place afin d'échanger et de collecter les différents types d'informations discutés dans la section précédente.

7.2.1 Description formelle du solveur SATHYS

Comme expliqué précédemment, notre approche SATHYS est basée sur la recherche locale. Le processus de recherche consiste à se déplacer d'une interprétation complète à une interprétation complète voisine jusqu'à obtenir une solution. À chaque étape, le nombre de clauses falsifiées tente d'être réduit (phase de descente). Lorsqu'aucune descente n'est possible, un minimum local est atteint et dans ce cas deux possibilités sont à considérer :

1. soit un critère d'échappement classique est utilisé (voir section 2.2) ;
2. soit la partie CDCL est appelée. Un littéral non affecté appartenant à l'interprétation complète utilisée par la recherche locale est sélectionné. Ce littéral est ensuite utilisé comme point de choix et la propagation unitaire est effectuée.

Dans les deux cas, l'ensemble des interprétations voisines de l'interprétation courante a changé et la partie recherche locale de notre hybridation peut de nouveau être exécutée. En effet, dans le premier cas, cette modification est contrôlée de manière classique à l'aide du critère d'échappement. Dans le second cas, puisque les variables fixées par la partie CDCL ne peuvent plus être flippées par la partie recherche locale, le voisinage direct de l'interprétation courante est modifié. De plus, la partie CDCL peut conduire à un conflit. Dans ce cas, une analyse de conflits est effectuée, une clause est apprise et un *backjumping* est réalisé. Lorsque cela arrive, certaines variables redeviennent libres et peuvent de nouveau être flippées.

Bien que le cœur de notre solveur hybride SATHYS soit une recherche locale, l'intégration du solveur CDCL conduit à rendre ce solveur capable de répondre à l'insatisfaisabilité d'une formule.

L'algorithme 7.1 décrit en détail cette approche. Il prend en entrée une formule CNF Σ et retourne `vrai` si le problème est montré cohérent et `faux` sinon. Quatre variables sont utilisées. Une interprétation partielle $\mathcal{I}_p$ pour la partie CDCL qui est initialisée avec les littéraux unitaires obtenus après la propagation unitaire sur la formule Σ initiale, une interprétation complète $\mathcal{I}_c$ pour la partie recherche locale laquelle est initialisée à l'aide la fonction `interprétationComplètePU` (voir paragraphe 7.2.2) en étendant l'interprétation partielle $\mathcal{I}_p$, Δ l'ensemble des clauses apprises et dl le niveau de décision courant (lignes 1–6). Après cette étape d'initialisation la partie recherche locale est exécutée. Si l'interprétation complète courante est un modèle de $\Sigma_{\mathcal{I}_p}$ alors SATHYS termine et retourne `vrai` (ligne 8). Sinon, s'il existe une interprétation voisine $\mathcal{I}'_c$ de $\mathcal{I}_c$ permettant de diminuer le nombre de clauses falsifiées, celle-ci devient l'interprétation courante (ligne 10). Il est important de noter que le voisinage de $\mathcal{I}_c$ est limité aux variables n'apparaissant pas dans $\mathcal{I}_p$ ($\mathcal{I}'_c \in \mathcal{N}(\mathcal{I}_c, \mathcal{V}_{\mathcal{I}_c \setminus \mathcal{I}_p})$). En effet, comme évoqué précédemment les variables utilisées dans $\mathcal{I}_p$ sont considérées comme tabou pour la partie recherche locale. Cependant, cette interprétation partielle évolue pendant la recherche et est modifiée par la partie CDCL de l'algorithme. Néanmoins à chaque tour de boucle (ligne 7) de notre méthode, nous avons $\mathcal{I}_p \subseteq \mathcal{I}_c$.

Lorsqu'aucune interprétation voisine ne permet une descente, un minimum local est atteint (ligne 11). Dans ce cas le prédicat `slsProgress` (voir paragraphe 7.2.4) est évalué. S'il retourne `vrai` alors un critère d'échappement classique est utilisé afin de sortir du minimum local (lignes 12–13). Sinon la partie CDCL de notre solveur est appelée (lignes 15–27) afin de modifier l'interprétation partielle.

Chaque fois que la partie CDCL est appelée, un littéral ℓ non affecté est choisi parmi l'ensemble des littéraux de l'interprétation complète $\mathcal{I}_c$ (fonction `heuristiqueLitéralDeBranchement`). Le niveau de décision courant est incrémenté, le littéral ℓ est affecté et la propagation unitaire est exécutée (lignes 15–18). Si l'interprétation partielle $\mathcal{I}_p$ obtenue par propagation conduit à la clause vide (ligne 19), une analyse de conflits classique (basée sur la notion de *first*-UIP) est effectuée et une clause γ est apprise. Si cette dernière est la clause vide alors le problème est montré inconsistant et `faux` est retourné (ligne 21). Sinon la clause assertive est ajoutée à l'ensemble des clauses apprises Δ, un retour arrière est effectué, le niveau de décision devient égal au niveau de *backjump* et la propagation unitaire est de nouveau réalisée (lignes 20–25). Ces opérations sont effectuées tant que l'interprétation partielle $\mathcal{I}_p$ est inconsistante (ligne 19). À la fin du processus, l'interprétation complète utilisée par la recherche locale est remise à jour grâce à l'interprétation partielle telle que $\mathcal{I}_p \subseteq \mathcal{I}_c$ (ligne 28). Comme pour les solveurs CDCL classique la base de clauses apprises est nettoyée périodiquement [22] (lignes 26–27).

22. Nous utilisons pour effectuer cette opération la même stratégie que celle utilisée dans le solveur MINISAT (voir section 3.1.5.2).

Ce processus est répété tant que le prédicat `redémarrage` n'est pas vérifié (ligne 6). Après quoi, tant qu'une solution n'a pas été trouvée (ou une preuve de l'insatisfiabilité de la formule) le solveur redémarre le processus précédent avec une nouvelle interprétation complète initiale.

Algorithme 7.1 : SATHYS

Données : Σ une formule CNF
Résultat : `vrai` si la formule Σ est satisfiable et `faux` si la formule est prouvé insatisfiable

```
1  dl ⟵ 0;                                              /* niveau de décision */
2  Δ ← ∅;                                  /* l'ensemble des clauses apprises */
3  I_p ⟵ ∅;                                     /* interprétation partielle */
4  α ⟵ propagationUnitaire(Σ, I_p);
5  si (α ≠ null) alors retourner faux ;                 /* problème inconsistant */
6  I_c ← I_p ∪ interprétationComplètePU(Σ_|I_p);
7  tant que (vrai ) faire
8      si (I_c ⊨ Σ) alors return SAT;
9      si ∃I'_c ∈ N(I_c, V_{I_c \ I_p}) telle que diff(Σ, I_c, I'_c) > 0 alors
10         I_c ⟵ I'_c;                                          /* descente */
11     sinon
           /* minimum local                                                */
12         si slsProgress() alors
               /* Comportement classique de la recherche locale            */
13             I_c ⟵ interprétation choisie suivant un critère d'échappement;
14         sinon
15             ℓ ⟵ heuristiqueLitéralDeBranchement(I_c \ I_p);
16             dl ⟵ dl + 1;
17             I_p ⟵ I_p ∪ {(ℓ^dl)};
18             α ⟵ propagationUnitaire(Σ, I_p);
19             tant que (α ≠ null) faire
20                 β ⟵ analyseConflit(Σ ∪ Δ, I_p, α);
21                 si (β = ⊥) alors retourner faux ;
22                 bl ⟵ calculRetourArrière(I_p, bl);
23                 Δ ⟵ Δ ∪ β;
24                 retourArrière(Σ ∪ Δ, I_p, bl);          /* mise à jour de I_p */
                   /* propagation du littéral assertif                         */
25                 α ⟵ propagationUnitaire(Σ, I_p)
26             si faireReduction() alors
27                 reductionClausesApprises(Δ);
28             I_c ← I_p ∪ I_c \ I_p;
29     si (redémarrage()) alors
30         propagationUnitaire(Σ ∪ Δ, I_p);
31         I_c ← I_p ∪ interprétationComplètePU(Σ_|I_p) ;
```

Contrairement aux stratégies de redémarrages utilisées habituellement dans les solveurs de recherche locale et qui dépendent du nombre de réparations ($maxReparations$ dans les algorithmes 2.2 et 2.3), le solveur SATHYS, comme les solveurs SAT modernes, s'appuie sur le nombre de conflits obtenu par la partie CDCL afin de déterminer s'il faut effectuer un redémarrage. Pour notre solveur nous avons choisi

d'utiliser une stratégie de redémarrage basée sur la suite de Luby avec $u = 100$ (voir section 3.1.5.3). Ce choix n'est pas anodin, puisqu'au vue des caractéristiques théoriques de celle-ci (logarithmiquement optimale lorsqu'aucune information sur le problème n'est fournie), elle permet de ratisser un plus large panel de problèmes.

En ce qui concerne le prédicat `slsProgress`, les fonctions `interprétationComplètePU` et `heuristiqueLitéralDeBranchement` laissés en suspens, nous les détaillons dans les parties suivantes.

7.2.2 Interprétation complète générée à partie de la propagation unitaire

Une des particularité des problèmes de la catégorie application est qu'ils sont pour la plupart composés d'un nombre important de clauses. Par conséquent la valeur de la fonction d'évaluation calculée sur une interprétation complète quelconque peut être très élevée (beaucoup de clauses falsifiées). Dans ce cas, considérer une interprétation complète générée de manière aléatoire peut amener la recherche locale à effectuer beaucoup de réparations avant d'atteindre un premier minimum local. Pour éviter cela, nous proposons d'utiliser une méthode gloutonne basée sur la propagation unitaire afin d'initialiser l'interprétation complète utilisée par la partie recherche locale de notre algorithme. Cette méthode, décrite dans l'algorithme 7.2, prend en entrée une formule CNF Σ et retourne une interprétation complète construite sur $\mathcal{V}_\Sigma$. La procédure commence par faire une sauvegarde de Σ dans Σ' et par initialiser l'interprétation complète $\mathcal{I}_c$ à vide. Une fois ces opérations effectuées, tant que l'interprétation $\mathcal{I}_c$ n'est pas complète, un littéral x est choisi parmi les littéraux de Σ' afin d'étendre $\mathcal{I}_c$ (ligne 4). Une fois ce dernier choisi, l'ensemble des littéraux obtenu par propagation unitaire à partir de Σ' est collecté dans l'ensemble $\mathcal{P}$ (ligne 5). Ensuite, une occurrence de chaque littéral de $\mathcal{P}$ apparaissant à la fois positivement et négativement est supprimée (ligne 6–8). L'interprétation $\mathcal{I}_c$ est alors étendu à l'aide des littéraux de $\mathcal{P}$ (ligne 9) et la formule Σ' est simplifiée par l'ensemble $\mathcal{P}$ (ligne 10).

Algorithme 7.2 : interprétationComplètePU

Données : Σ une formule CNF
Résultat : $\mathcal{I}_c$ une interprétation complète de Σ

```
1  Σ' ← Σ;
2  I_c ← ∅;
3  tant que |I_c| ≠ |V_Σ| faire
4      I_c ← I_c ∪ {x | x ∈ V_Σ \ I_c};
5      P ← {x} ∪ {y | Σ'_|x ⊨* y};
6      pour chaque y ∈ P faire
7          si ȳ ∈ I_c alors
8              P ← P \ {y};
9      I_c ← I_c ∪ P;
10     Σ' ← Σ'_|P;
11 retourner I_c;
```

7.2.3 L'heuristique de branchement

L'heuristique de branchement utilisée par la partie CDCL du solveur SATHYS est déterminée grâce à la fonction `heuristiqueLitéralDeBranchement`. Cette dernière dépend d'une heuristique de

choix de variables et d'une l'heuristique de choix de polarité. Dans le solveur SATHYS ces heuristiques sont définies de la manière suivante :

- l'heuristique de choix de variable utilisée est une version modifiée de l'heuristique VSIDS introduite dans la section 3.1.4.3. Cette méthode, comme pour la méthode de base, augmente le poids des variables en cause dans les conflits lorsque l'extraction d'un *nogood* est effectuée (analyse de conflits). De plus, lors de la phase de recherche locale et lorsqu'un minimum local est atteint, l'interprétation complète courante est examinée afin de détecter d'éventuelles variables frontières. Lorsque de telles variables sont identifiées, leur poids est augmenté. De cette manière, les variables frontières sont susceptibles d'être sélectionnées et par conséquent d'être utilisées pour diriger la recherche vers une partie difficile du problème ;
- l'heuristique de choix de polarité est quant à elle basée sur la notion de *progress saving* introduite par Pipatsrisawat et Darwiche (2009b). Cependant, contrairement à cette dernière, l'interprétation complète $\mathcal{P}$ utilisée lors de la sélection de la phase de la variable (voir 3.1.4.4) est modifiée à chaque redémarrage. Plus précisément, à chaque redémarrage l'interprétation $\mathcal{P}$ est remplacée par la meilleure interprétation complète (meilleure en terme de nombre de clauses falsifiées) obtenue par le processus de recherche locale. De cette manière, la partie CDCL de notre solveur est guidée dans un espace de recherche espéré prometteur.

7.2.4 Le prédicat `slsProgress`

Les appels aux différentes parties (CDLS et recherche locale) de notre solveur sont contrôlés par le prédicat `slsProgress` (voir l'algorithme 7.1, ligne 12). Une mauvaise définition de ce dernier peut conduire à privilégier une méthode au détriment de l'autre. En effet, un prédicat toujours évalué à `vrai` rend notre solveur identique à une méthode de recherche locale classique. Au contraire, un prédicat toujours évalué à `faux` favorise énormément la partie CDCL ce qui, comme le montre les expérimentations reportées dans la partie 7.3.2, rend notre approche inefficace sur les problèmes aléatoires. Afin de pallier ce problème, et donc d'estimer convenablement quelle partie de notre algorithme il faut privilégier, nous nous sommes appuyés sur les travaux de Hoos (2002) introduit dans la section 2.3. Plus précisément, pour une formule CNF Σ, nous considérons une variable entière $progress$ qui est augmentée lorsque la recherche locale progresse dans sa recherche de solution ou lorsque la partie CDCL est considérée être en situation d'échec. Nous avons caractérisé trois situations dans lesquels il nous semble intéressant d'augmenter la valeur de $progress$. Ces situations sont :

- lorsqu'un redémarrage est effectué. En effet, un redémarrage peut être assimilé à échec de la partie CDCL dans sa recherche de solution. Dans ce cas, nous choisissons de réinitialiser $progress$ avec le résultat de l'opération suivant : $Luby(100) \times Nombre\ de\ redémarrages$;
- lorsque la partie CDCL atteint un conflit. Dans ce cas, l'interprétation partielle est dans la plupart des cas réduite. Il semble donc intéressant de laisser à la recherche locale l'opportunité d'explorer l'espace de recherche qui était préalablement fixé. Pour cela, nous ajoutons $|\mathcal{I}_p| - |\mathcal{I}'_p|$ à $progress$ tel que $\mathcal{I}_p$ et $\mathcal{I}'_p$ sont respectivement l'interprétation partielle obtenue avant et après le retour arrière non chronologique ;
- lorsque la recherche locale atteint une interprétation prometteuse. Pour cela, nous initialisons une variable $maxSatRestart$ qui correspond au nombre de clauses falsifiées par l'interprétation complète générée juste après chaque redémarrage. Puis lorsque la recherche locale permet d'obtenir une interprétation telle que le nombre de clauses falsifiées par celle-ci est inférieur à $maxSatRestart$, alors $maxSatRestart$ est mis-à-jour avec cette valeur et $progress$ est augmenté de 1000.

Afin de déterminer si la recherche locale échoue dans sa recherche de solution, nous avons choisi de décrémenter $progress$ chaque fois qu'un minimum local est atteint. En effet, un minimum local peut

être assimilé à une situation d'échec pour la recherche locale et par conséquent la partie CDCL doit être privilégiée.

Le prédicat `slsProgress` est alors évalué simplement puisqu'il retourne `vrai` si $progress$ est supérieur à 0 et `faux` sinon.

7.3 Études expérimentales

Dans cette section nous étudions expérimentalement notre solveur hybride SATHYS. Cette étude est divisée en trois parties. Dans la première, nous évaluons les performances de notre méthode avec plusieurs stratégies d'échappement. Dans la seconde, nous étudions l'apport des informations échangées par les parties recherche locale et CDCL. Dans la dernière, nous comparons notre méthode avec les solveurs de l'état de l'art.

L'ensemble des résultats expérimentaux reportés dans cette section a été obtenu sur un Xeon 3.2 GHz avec 2 GByte de RAM. Le temps CPU a été limité à 1200 secondes. Les instances utilisées sont issues de la compétitions SAT 2009 (Le Berre et Roussel 2009). Elles sont divisées en trois catégories : *crafted* (281 instances), application (292) et aléatoire (570). Toutes ces instances sont prétraitées par SATELITE (Eén et Biere 2005).

7.3.1 Choix de la stratégie d'échappement utilisée dans SATHYS

Pour commencer notre étude expérimentale, nous étudions les performances de notre solveur vis-à-vis de trois stratégies d'échappement : WALKSAT (Selman *et al.* 1994), NOVELTY et RNOVELTY (McAllester *et al.* 1997). Les trois approches ainsi définies sont nommées respectivement SATHYS (WALKSAT), SATHYS (NOVELTY) et SATHYS (RNOVELTY) et leurs résultats sont reportés dans le tableau 7.1. Nous pouvons observer sur ce dernier que sur les instances structurées (*crafted* et application) les performances de SATHYS sont quasi similaires quelque soit la métaheuristique employée. Ces résultats ne sont pas surprenants puisque sur ce type d'instances les différents critères d'échappement obtiennent des performances quasiment identiques. En ce qui concerne les résultats obtenus sur la catégorie d'instances aléatoires, nous remarquons que la sélection du critère d'échappement est critique. En effet, puisqu'il conditionne les performances d'un solveur de recherche locale, il n'est pas inattendu d'observer que le nombre d'instances résolues par les méthodes utilisant les critères d'échappement NOVELTY et RNOVELTY soit supérieur au nombre d'instances résolues par la méthode basée sur la métaheuristique WALKSAT.

	Crafted			Application			Aléatoire			
	total	(sat	unsat)	total	(sat	unsat)	total	(sat	unsat)	total
SATHYS (WALKSAT)	100	(70	30)	140	(60	80)	153	(153	0)	393
SATHYS (NOVELTY)	**107**	(**74**	**33**)	140	(60	80)	**190**	(**190**	0)	437
SATHYS (RNOVELTY)	104	(71	**33**)	**148**	(**63**	**85**)	189	(189	0)	**441**

TABLE 7.1 – Étude des performances du solveur SATHYS pour les stratégies d'échappement WALKSAT, NOVELTY et RNOVELTY. Pour chaque catégorie et pour chaque solveur, nous reportons le nombre d'instances satisfaisables, insatisfaisables et totales résolues.

Dans la suite de ce chapitre, nous nommons SATHYS la version de notre solveur qui utilise comme critère d'échappement la stratégie RNOVELTY (SATHYS (RNOVELTY)).

7.3.2 Étude de l'apport des informations échangées

Dans cette section, nous évaluons l'apport des informations échangées sur l'efficacité de notre solveur. Pour cela, nous comparons SATHYS avec les variantes suivantes :

- SATHYS$_{nb}$: l'heuristique de choix de variables n'est pas mise à jour lorsqu'une nouvelle variable frontière est découverte ;
- SATHYS$_{cdclAtEachLM}$: le prédicat `slsProgress` utilisé est toujours évalué à `faux`. Dans ce cas, la partie CDCL est appelée à chaque minimum local ;
- SATHYS$_{noPolarity}$: l'heuristique de choix de polarité n'est pas remise-à-jour lorsque la recherche locale atteint une nouvelle valeur de MAXSAT.

Le tableau 7.2 résume les résultats obtenus par les quatre versions de notre solveur SATHYS.

Solveur	Crafted	Application	Aléatoire	Total
SATHYS	104	**148**	189	**441**
SATHYS$_{nb}$	103	144	**191**	438
SATHYS$_{cdclAtEachLM}$	101	141	8	250
SATHYS$_{noPolarity}$	**106**	142	188	436

TABLE 7.2 – Étude de l'apport de l'échange d'informations dans le solveur SATHYS. Pour chaque catégorie de problème, nous reportons le nombre d'instances résolues par chacune des versions de SATHYS. Nous donnons aussi le nombre d'instances total résolues par chacune des méthodes.

Avant d'analyser plus finement la table des résultats (tableau 7.2), remarquons que la version de SATHYS qui incorpore toutes les informations permet de résoudre globalement plus d'instances. Par conséquent, l'échange d'informations tel que nous l'avons défini dans notre solveur permet d'augmenter la robustesse et l'efficacité de notre solveur.

Concernant les résultats obtenus par SATHYS$_{nb}$, nous pouvons noter que le réajustement de l'heuristique de choix de variables avec les variables frontières ne permet pas d'améliorer les performances de notre méthode sur les instances aléatoires. Ce résultat n'est pas surprenant puisque sur ce type d'instances l'heuristique de choix de variables utilisée par la partie CDCL n'a pas de réel impact sur les performances de la méthode. Par conséquence, ajuster cette dernière n'est pas nécessaire et est au contraire consommatrice de temps. Pour ce qui est des catégories *crafted* et application, nous pouvoir voir que focaliser la partie CDCL sur les variables frontières permet d'améliorer l'efficacité de la méthode.

En ce qui concerne la version SATHYS$_{cdclAtEachLM}$, nous pouvons voir que les résultats obtenus sur les instances aléatoires sont très mauvais. Ce comportement peut être expliqué par le fait que les algorithmes CDCL sont mauvais sur ce type d'instances et qu'appeler la partie CDCL à chaque minimum local privilégie trop la partie complète de notre algorithme. Nous pouvons noter que le nombre d'appels à la partie CDCL a un impact important sur les performances de SATHYS. Par conséquent, évaluer correctement si la recherche locale progresse (prédicat `slsProgress`) est crucial pour l'efficacité de notre approche. Pour terminer remarquons que, malgré le fait que la partie CDCL soit privilégiée, le nombre d'instances résolues dans les catégories *crafted* et application est toujours inférieur au nombre d'instances résolues dans le cas où la recherche locale est appelée plus fréquemment.

Quant aux résultats obtenus par SATHYS$_{noPolarity}$, nous pouvons voir que cette version obtient de bons résultats sur la catégorie d'instances *crafted*. Cependant, cette approche est moins compétitive sur les instances de la catégorie application. Sur ces instances, ne pas appliquer la mise à jour de l'heuristique de choix de polarité à chaque redémarrage dégrade les performances.

7.3.3 Étude comparative

Dans cette section nous comparons SATHYS avec les solveurs ayant obtenus une distinction lors de la compétition SAT qui s'est déroulée en 2009 (cinq de ces solveurs ont obtenu une médaille d'or à cette compétition et deux de ces solveurs font partie des meilleurs solveurs hybrides). Plus précisément, notre approche a été comparée avec :

- Deux méthodes de recherche locale :
  1. ADAPTG^{2}WSAT (Li *et al.* 2007) ;
  2. GNOVELTY+ (Pham et Gretton 2009) ;
- Deux approches hybrides :
  1. HYBRIDGM (Balint *et al.* 2009)
  2. HINOTOS (Letombe et Marques-Silva 2008).
- Cinq approches complètes :
  1. MINISAT (Sörensson et Eén 2009) ;
  2. GLUCOSE (Audemard et Simon 2009a) ;
  3. PRECOSAT (Biere 2008b) ;
  4. RSAT (Pipatsrisawat et Darwiche 2007) ;
  5. CLASP (Gebser *et al.* 2007).
- Un solveur portfolio (séquentiel) : SATZILLA_I (Xu *et al.* 2008).

Le tableau 7.3 résume, sur les trois catégories d'instances de la compétition SAT 2009, les résultats de SATHYS et des solveurs énumérés précédemment.

	Crafted			Application			Aléatoire			
	total	(sat	unsat)	total	(sat	unsat)	total	(sat	unsat)	total
ADAPTG^{2}WSAT	68	(68	0)	8	(8	0)	**294**	(**294**	0)	375
GNOVELTY+	54	(54	0)	7	(7	0)	281	(281	0)	342
SATHYS	104	(71	33)	148	(63	85)	189	(189	0)	**441**
HYBRIDGM	56	(51	5)	5	(5	0)	**294**	(**294**	0)	355
HINOTOS	105	(69	36)	107	(39	68)	77	(65	11)	288
MINISAT	99	(72	27)	152	(59	93)	3	(3	0)	254
GLUCOSE	114	(75	39)	152	(54	98)	17	(17	0)	266
PRECOSAT	122	(81	41)	**164**	(**65**	**99**)	2	(2	0)	288
RSAT	105	(71	34)	143	(53	90)	5	(5	0)	253
CLASP	**131**	(78	**53**)	138	(53	85)	84	(66	18)	353
SATZILLA_I	128	(**86**	42)	142	(60	82)	145	(90	**55**)	415

TABLE 7.3 – Comparaison du solveur SATHYS vis-à-vis d'un ensemble de solveurs de l'état de l'art. Pour chaque catégorie de problème et pour chaque solveur, nous reportons le nombre d'instances satisfaisables, insatisfaisables et totales résolues.

Commençons cette étude en comparant les performances des solveurs de recherche locale par rapport à celle de notre solveur SATHYS. Tout d'abord, notons que ces méthodes ne sont performantes que sur les instances de la catégorie aléatoire. Ce résultat n'est pas surprenant puisque ces solveurs sont uniquement dédiés à ce type de problèmes. Sur les autres catégories d'instances, les méthodes à base de

recherche locale obtiennent généralement de très mauvais résultats. Le manque d'efficacité de ce type d'approche sur les problèmes structurés a même été pointé du doigt lors de la compétition SAT de 2009 (voir présentation de la compétition SAT 2009 (Le Berre *et al.* 2009)). Par conséquent, l'amélioration de ces dernières sur ce type d'instances est devenu un véritable challenge. Puisque notre solveur peut être vu comme une méthode de recherche locale qui utilise un solveur CDCL pour sortir des minima locaux, les bonnes performances qu'il obtient sur les instances des catégories *crafted* et application permet d'une certaine manière d'apporter une réponse à ce dernier challenge.

En ce qui concerne les performances des solveurs hybrides par rapport à notre solveur, nous voyons clairement (dans le tableau 7.3) que notre approche est beaucoup plus robuste. En effet sur les instances de la catégorie application, SATHYS permet de résoudre approximativement 33% d'instances de plus que le solveur HINOTOS et 100% d'instances de plus que HYBRIDGM. Pour ce qui est des instances de la catégorie aléatoire, l'approche hybride qui obtient les meilleurs performances est HYBRIDGM (environ 50% d'instances résolues en plus). Ceci n'est pas surprenant puisque ce dernier a été développé dans l'optique de résoudre de telles instances. Néanmoins, remarquons que notre solveur obtient tout de même des résultats très satisfaisants (SATHYS résout environ 2,5 fois plus d'instances que HINOTOS). En ce qui concerne les instances de la catégorie *crafted*, les performances de notre solveur sont identiques à ceux de HINOTOS et sont bien supérieures à ceux de HYBRIDGM (environ 50% d'instances résolues en plus).

Si nous analysons les résultats obtenus par les solveurs complets, nous remarquons que notre solveur est très compétitif sur les instances des catégories *crafted* et application. En effet, SATHYS obtient dans sur ces catégories de problèmes des résultats très honorables, là où les autres approches reposant sur la recherche locale sont à la peine. Sur les instances de la catégorie aléatoire, il n'est pas surprenant de constater que notre approche est bien meilleure, puisque les solveurs CDCL sont très mauvais sur les problèmes de ce type.

Un des résultats les plus surprenant est que notre solveur SATHYS est plus robuste que le solveur portfolio SATZILLA. En effet, bien que ce dernier soit composé des meilleurs solveurs sur chacune des catégorie de problème, notre solveur résout légèrement plus d'instances (environ 6% d'instances résolues en plus). Ceci est surprenant, puisque le principe de ce solveur est, grâce à une analyse de la structure du problème, d'identifier la nature du problème et de lancer le solveur approprié.

Pour résumer les résultats reportés dans le tableau 7.3, nous pouvons dire que, si nous considérons les trois catégories de problème (*crafted*, application et aléatoire), SATHYS est le solveur le plus compétitif. Il est aussi le plus robuste puisqu'il résout globalement plus d'instances que tous les autres solveurs. De plus, nous pouvons noter que pour chacune des catégories notre solveur offre de bonnes performances.

Le tableau précédent permet de fournir des informations quant au nombre d'instances résolues par chacune des approches. Afin d'obtenir une comparaison au niveau des temps de résolution, nous proposons de comparer les différents solveurs à l'aide de plusieurs nuages de points. Pour cela, nous reportons sur les figures 7.3, 7.4, 7.5 et 7.6, les nuages de points relatifs aux comparaisons effectuées entre SATHYS et respectivement PRECOSAT (le solveur de référence sur la catégorie d'instances application), HINOTOS (un des solveurs hybrides les plus connus), CLASP (le vainqueur de la compétition SAT 2009 dans la catégorie *crafted*) et SATZILLA (le solveur qui a obtenu les meilleures performances après SATHYS).

Sur toutes ces figures, chaque point représente le temps mis par chacune des deux méthodes pour résoudre une instance Π. L'axe des abscisses correspond alors au temps (échelle logarithmique) mis par le solveur SATHYS pour résoudre Π et l'axe des ordonnées le temps (échelle logarithmique) mis par les autres solveurs pour résoudre ce même problème. Chaque point en dessous de la diagonale représente donc une instance pour lequel SATHYS est plus rapide.

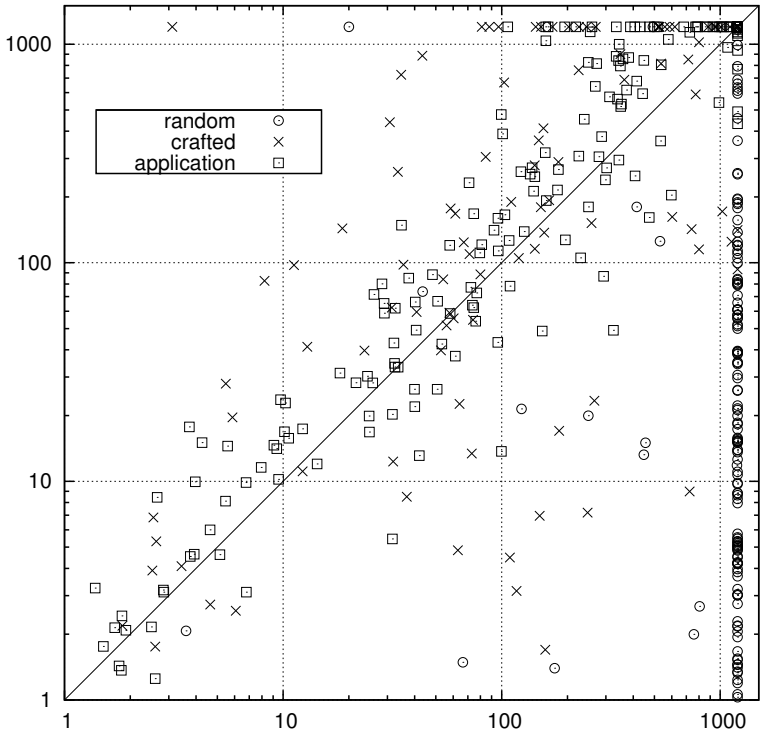


FIGURE 7.3 – Nuage de points : SATHYS versus PRECOSAT.

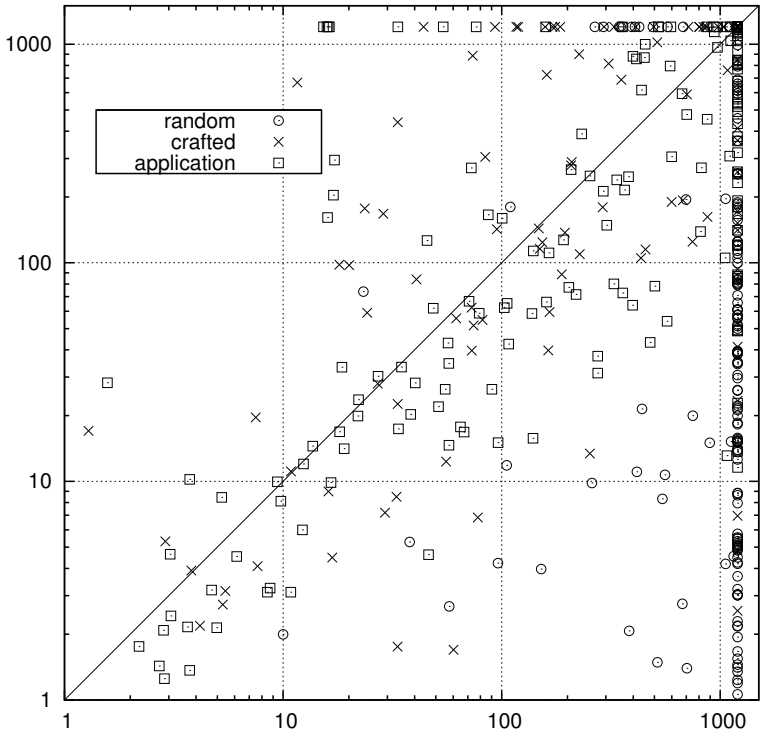


FIGURE 7.4 – Nuage de points : SATHYS versus HINOTOS.

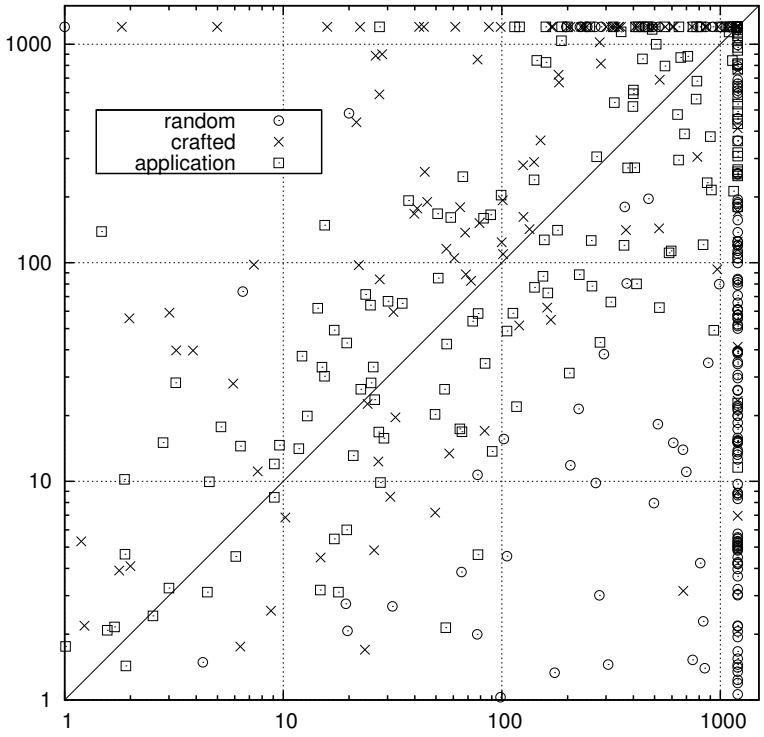


FIGURE 7.5 – Nuage de points : SATHYS versus CLASP.

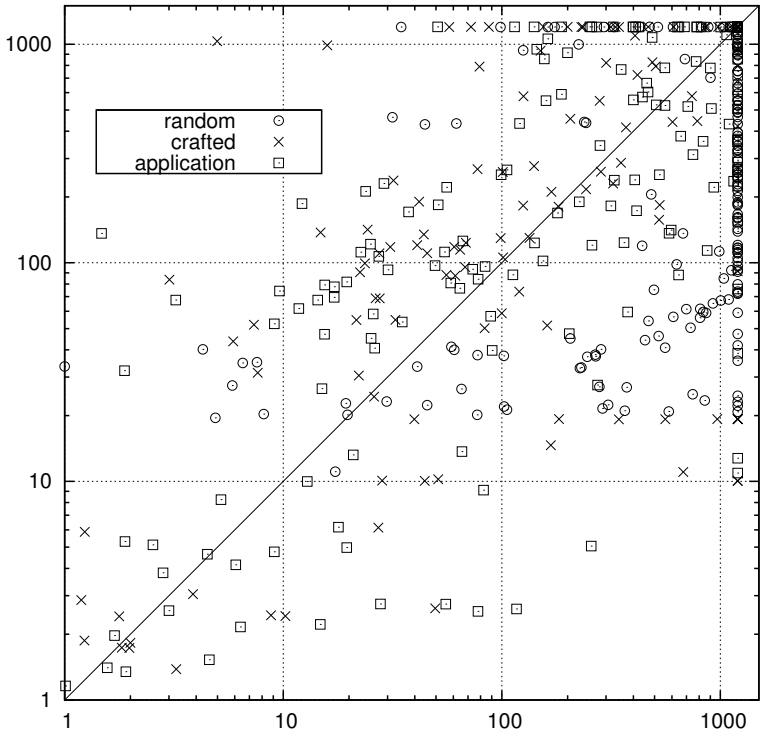


FIGURE 7.6 – Nuage de points : SATHYS versus SATZILLA

L'analyse des différents nuages de points permet tout d'abord de conclure que notre solveur est plus efficace que HINOTOS. En effet, la plupart des points reportés dans la figure 7.4 se trouve en dessous de la diagonale.

En ce qui concerne la comparaison avec PRECOSAT (lequel est l'un des solveurs CDCL les plus rapides), nous pouvons voir sur la figure 7.3 que, bien que notre solveur semble plus lent, SATHYS résout globalement plus d'instances. Concernant le solveur CLASP, nous pouvons voir sur la figure 7.5 que notre solveur résout de manière similaire les instances des catégories *crafted* et application. Néanmoins, nous remarquons que notre solveur est tout de même beaucoup plus performant que ce dernier sur les instances aléatoires. Finalement, concernant la comparaison des performances de SATHYS et de SATZILLA, il est très difficile à la vue de la figure 7.6 de déterminer le solveur le plus efficace. En effet, la répartition des points de part et d'autre de la diagonale ne permet aucune conclusion.

Nous terminons cette étude comparative en présentant les temps obtenus par notre solveur sur un ensemble d'instances des catégories *crafted* et application. Bien que ces instances soient difficiles, nous pouvons voir sur le tableau 7.4 que notre solveur permet de résoudre des instances satisfiables et insatisfiables. Afin d'être équitable nous avons aussi, pour chacune des instances inscrites dans la colonne *best* le temps d'un des 50 solveurs de la compétition SAT 2009, qui pour une instance donnée, est le plus rapide.

7.3.4 Synthèse

L'ensemble des expérimentations reporté dans cette section a permis d'évaluer les performances de notre approche hybride sur un large panel d'instances. Ces instances, issues de la compétition SAT 2009, ont été sélectionnées de manière à représenter le plus large échantillon possible. Ainsi, les résultats reportées dans cette section sont représentatifs du comportement de notre solveur quelque soit le type d'instances considérées. Cette étude a été divisée en trois parties.

Dans la première, nous avons pu constater que la méthode de recherche locale utilisée au sein de notre méthode influence énormément les performances de notre solveur sur les instances de la catégorie aléatoire. En effet, un mauvais choix de stratégie d'échappement conduit à dégrader fortement les performances du solveur sur ce type de problème (perte de performances d'environ 25%).

Nous avons ensuite évalué l'apport du partage d'informations entre la partie recherche locale et la partie CDCL de notre solveur. Les résultats que nous avons obtenus nous permet d'affirmer que ces dernières sont importantes et qu'elles permettent de résoudre plus efficacement les instances et cela quelque soit la catégorie. Dans cette étude, nous avons aussi pu mesurer l'importance du prédicat *slsProgress* sur les performances de SATHYS. Nous avons pu constater qu'un mauvais choix de celui-ci entraîne une dégradation des performances (environ 75% d'instances résolues en moins).

Pour terminer, l'étude comparative que nous avons conduite a permis de mettre en exergue le fait que SATHYS est très efficace dans les trois catégories d'instances (*crafted*, application et aléatoire) et qu'il peut, par conséquent, être considéré comme un des solveurs les plus robustes. De plus, à partir des résultats reportés dans cette étude, nous pouvons aussi considérer que SATHYS est le premier solveur hybride à réellement proposer une réponse à trois des dix challenges proposés par Selman *et al.* (1997) :

- Challenge 5 : *Design a practical stochastic local search procedure for proving unsatisfiability* ;
- Challenge 6 : *Improve stochastic local search on structured problems by efficiently handling variable dependencies* ;
- Challenge 7 : *Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.*

	SAT	BEST	SATHYS	HINOTOS	GLUCOSE	PRECOSAT	CLASP	SATZILLA
q_query_3_L100_coli	N	183	677	–	577	**414**	780	–
post-c32s-col400-16	N	66	247	380	714	141	**66**	125
countbitsarray02_32	N	926	1100	–	**926**	–	–	–
maxxororand032	N	579	**768**	–	–	–	–	–
minand128	N	16	71	219	26	26	**23**	212
rpoc_xits_08_UNSAT	N	154	1036	1115	398	**159**	–	589
gt-ordering-unsat-gt-060	N	15	171	–	–	**149**	–	–
9dlx_vliw_at_b_iq3	N	430	1137	–	–	–	1095	**430**
gss-19-s100	Y	38	641	–	611	–	–	**38**
UCG-20-5p1	Y	413	835	–	–	**537**	–	–
UTI-15-10p1	Y	392	935	–	**392**	–	–	1140
gt-ordering-sat-gt-040	Y	15	**6**	–	–	149	–	–
em_9_3_5_exp	Y	18	288	209	**51**	181	140	276
ndhf_xits_20	Y	1.6	180	–	–	249	–	75
partial-10-15-s	Y	228	**1092**	–	–	–	–	–
vmpc_30	Y	18	825	–	–	292	**159**	551
velev-pipe-sat-1.0-b7	Y	14	294	–	–	343	–	**87**
new-difficult-21-168-19-90	Y	0.1	**141**	–	–	–	370	416
mod3block_3vars_9gates…	Y	5.7	**4.8**	–	126	63	26	24
sgen1-sat-160-100	Y	0.1	143	147	–	**18**	157	
rbsat-v945c61409g10	Y	21	362	–	898	**147**	150	933
instance_n8_i9_pp	Y	41	**115**	454	1117	800	–	421

TABLE 7.4 – Résultats obtenus par SATHYS sur un ensemble d'instances structurées. Les instances se trouvant au dessus de la ligne vertical sont des instances de la catégorie *crafted* tandis que ceux se trouvant sous cette dernière sont des instances de la catégorie application. Les temps reportés dans ce tableau sont exprimés en secondes.

7.4 Conclusion

Dans ce chapitre, nous avons introduit un nouveau schéma d'hybridation permettant de faire collaborer un solveur de recherche locale et un solveur CDCL. Cette combinaison se veut originale, puisque contrairement aux autres approches hybrides classiques aucune des deux méthodes ne peut être considérée comme esclave de l'autre. En effet, dans notre approche la partie recherche locale est utilisée afin de guider la partie CDCL vers les parties difficiles (insatisfiable) du problème tandis que l'approche CDCL peut être considérée comme un critère d'échappement pour la méthode de recherche locale. En ce qui concerne la partie expérimentations, notre solveur SATHYS obtient de bien meilleurs résultats que les autres solveurs hybrides. De plus, bien que légèrement moins bon que certains solveurs CDCL, notre solveur est très compétitif sur les problèmes structurés (catégories *crafted* et application). Le résultat le plus surprenant est sans nulle doute le fait que SATHYS soit également plus robuste que SATZILLA qui est un solveur portefolio. Surtout, SATHYS est le solveur le plus robuste et il peut être considéré comme le premier vrai succès dans les méthodes de recherche hybride.

Bien que les performances de SATHYS soient particulièrement remarquables, le développement de celui est très récent et peut être amélioré de plusieurs manières. La première consiste à renforcer l'échange d'informations entre les deux approches. Pour cela, plusieurs pistes peuvent être explorées. Premièrement, nous désirons proposer une politique dynamique de redémarrage pour notre solveur hybride SATHYS, en tenant compte des informations recueillies par le moteur de recherche locale. Il peut être aussi intéressant d'utiliser les informations récoltées par la recherche locale afin d'ajuster le poids des clauses de manière à définir une nouvelle politique de nettoyage de la base de clauses apprises. Une dernière piste sur laquelle nous envisageons de travailler est l'exploitation au sein de SATHYS des récents progrès faits sur le calcul de MUS. En effet, il s'avère que sur certaines instances, la méthode proposée par Grégoire *et al.* (2009a), et qui s'appuie sur une utilisation de la recherche locale afin d'identifier les clauses susceptibles d'appartenir à un MUS, permet de délivrer plus rapidement une approximation d'un MUS (donc de prouver l'inconsistance de la formule) que les meilleures approches CDCL à prouver que l'instance dans sa globalité n'admet pas de modèle. Nous envisageons donc, comme le fait la partie CDCL pour la recherche locale, d'utiliser la partie recherche locale pour réduire la formule sur laquelle travaille la partie CDCL.

Une autre manière d'améliorer les performances du solveur SATHYS consiste à améliorer séparément les deux approches qui le compose. Pour cela, nous souhaitons intégrer dans notre solveur d'autre méthodes de recherche locale (RSAPS, GNOVELTY+, ADAPTG^{2}WSAT, ...) ou encore exploiter les dernières avancées réalisées dans les solveurs CDCL.

Finalement, nous envisageons de développer le solveur SATHYS dans le contexte parallèle. En effet, nous constatons aujourd'hui une explosion du nombre de cœurs dans les processeurs et il est évident que les futurs solveurs devront exploiter l'ensemble de la puissance de calcul offerte par ces processeurs (voir chapitre 5).

Chapitre 8

Solveur hybride pour la résolution pratique de CSP

Sommaire

CE CHAPITRE montre que les techniques complètes et la recherche locale peuvent s'avérer également complémentaires pour permettre de résoudre le problème CSP. Il présente une combinaison synergique de la recherche locale et d'éléments de techniques complètes, qui souvent surpassent les approches complètes usuelles. Cette méthode n'est pas seulement complète : elle est aussi robuste dans le sens où elle peut aussi bien résoudre les instances satisfiables et insatisfiables, structurées ou aléatoires. En effet notre étude expérimentale montre qu'elle résout globalement plus d'instances que les techniques couramment utilisées.

Une des clés de notre approche est que la recherche locale permet d'extraire les informations nécessaires afin de guider autant que possible la recherche vers les sous-parties difficiles du problème CSP considéré. Cette idée, que nous avons déjà exploitée dans le chapitre précédent avec le solveur SATHYS, est ici raffinée grâce au concept original de variables FAC (*Falsified in All Constraints*). Ces dernières, dans le cadre CSP, sont des variables qui apparaissent dans toutes les contraintes violées pour une certaine affectation de l'ensemble des variables. Ces variables apparaissent donc dans au moins une contrainte de chaque noyau minimalement incohérent d'un CSP (voir section 1.3.2). De manière intéressante, la recherche locale permet souvent de détecter efficacement des variables FAC, et permet à une approche complète de type MAC de se focaliser en priorité sur ces variables. Ainsi, les résultats sont significativement améliorés sur de nombreuses instances. De même, des heuristiques puissantes (en particulier $dom/wdeg$ (Boussemart *et al.* 2004)) développées dans le cadre des approches complètes utilisées en CSP peuvent jouer un rôle essentiel dans le processus de la recherche locale. La méthode proposée, appelée FAC-SOLVER, correspond à la combinaison élaborée de la recherche locale et d'étapes issues de méthodes complètes. Ceci est permis par des échanges réciproques d'informations entre les différentes composantes mises en jeu.

Ce chapitre est organisé de la manière suivante. Dans la prochaine section, le concept de variables FAC est présenté. Ensuite, nous présentons et étudions expérimentalement le solveur de recherche locale qui a servi de base à notre approche hybride, l'architecture de cette dernière est détaillée dans la section 8.3. Avant de conclure et de donner quelques perspectives, l'ensemble des expérimentations conduites sur FAC-SOLVER est discuté.

Une partie de ces travaux a donné lieu à deux publications (Grégoire *et al.* 2011b;a)

8.1 Les variables FAC

Une des clés de l'efficacité de notre approche est liée au concept de variables FAC présenté ici.

Définition 8.1 (variable FAC)**.** *Soient un réseau de contraintes $\mathcal{P}$ et une interprétation complète $\mathcal{I}$. Une variable* FAC *est une variable apparaissant dans toutes les contraintes de $\mathcal{P}$ violées par $\mathcal{I}$.*

Exemple 8.1. *Considérons le réseau de contraintes $\mathcal{P}$ représenté dans la figure 8.1 et l'interprétation complète $\mathcal{I} = \{(X_1 = 1), (X_2 = 1), (X_3 = 1), (X_4 = 1)\}$. Nous pouvons voir sur cette figure que la variable X_1 est une variable* FAC*. En effet, l'ensemble des contraintes falsifiées par l'interprétation $\mathcal{I}$ est $\{C_1, C_5\}$. Puisque $X_1 \in var(C_1) \cap var(C_5)$ alors, par définition, nous avons que X_1 est une variable* FAC*.*

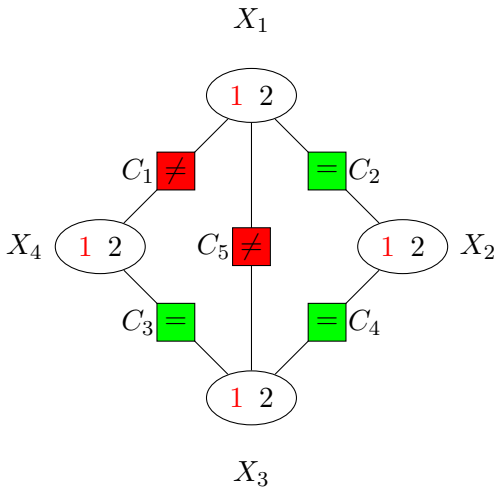


FIGURE 8.1 – Variable FAC associée à une interprétation complète d'un réseau de contraintes.

Ce concept peut être rapproché de la notion de point frontière introduite par Goldberg (2009) dans le cadre du problème SAT et présentée dans le chapitre précédent (voir 7.1.1). En effet, dans le cadre de SAT, une variable x est considérée comme frontière s'il existe une interprétation complète $\mathcal{I}$ construite sur l'ensemble des variables de la formule telle que x appartient à toutes les contraintes falsifiées par $\mathcal{I}$ (ce qui est typiquement la définition d'une variable FAC). Ici, nous avons choisi de ne pas nommer nos variables frontières en raison de la nuance suivante : dans le cadre du problème SAT, lorsqu'une variable x est détectée frontière pour l'interprétation $\mathcal{I}$, il est possible de satisfaire l'ensemble des contraintes falsifiées par $\mathcal{I}$ en inversant la valeur de vérité de la variable x. Cette propriété permet d'établir qu'une variable frontière est à la frontière entre la satisfiabilité et l'insatisfiabilité de la formule. Dans le cadre de CSP, lorsqu'une variable est détectée comme FAC, il n'est pas certain que modifier sa valeur satisfasse

la contrainte. Ainsi, la notion de frontière mise en avant par Goldberg ne peut être appliquée ici. C'est pour cela que nous avons choisi d'utiliser des termes différents.

Néanmoins, la propriété sur les variables frontières énoncées dans la partie précédente peut être étendue aux variables FAC pour la résolution du problème CSP.

Propriété 8.1. *Soit un réseau de contraintes $\mathcal{P}$ insatisfiable, si X est une variable* FAC *alors X apparaît dans au moins une contrainte de chaque* MUC *du $\mathcal{P}$.*

Preuve 8.1. *Quelle que soit l'interprétation complète $\mathcal{I}$, au moins une contrainte de chaque* MUC *est violée par $\mathcal{I}$ (par définition des* MUC*). Ainsi, si la variable X a été détectée* FAC *à l'aide de l'interprétation $\mathcal{I}$, alors par définition d'une variable* FAC*, X apparaît dans toutes les contraintes violées par $\mathcal{I}$. Donc X appartient à au moins une contrainte de chaque* MUC*.*

Puisqu'elles apparaissent dans tous les MUC, les variables FAC peuvent être considérées comme très intéressantes pour conduire à l'établissement de la preuve de l'incohérence d'un réseau de contraintes. Cependant, pour les mêmes raisons que pour les variables frontières, établir si une variable est FAC est calculatoirement difficile. En effet, vérifier si une contrainte donnée appartient à au moins un MUC est algorithmiquement aussi difficile que de déterminer si une clause appartient à au moins un MUS (Σ_2^p dans le pire des cas). De plus, un CSP peut posséder un nombre exponentiel de MUC. Par conséquent décider si une variable est FAC en calculant l'ensemble des MUC du problème est une fois de plus irréalisable dans le pire des cas. Pour pallier ce problème, nous proposons une nouvelle fois d'utiliser la recherche locale afin de détecter des variables FAC (voir section 8.2).

La propriété 8.2 met en avant le fait qu'il n'est pas toujours possible de trouver des variables FAC. Cependant cette situation est rare, puisque dans la plupart des cas les instances insatisfiables ne possèdent pas (ou très rarement) de noyaux inconsistants disjoints, synonyme de deux sources différentes d'incohérence.

Propriété 8.2. *Un réseau de contraintes $\mathcal{P}$ qui contient au moins deux* MUC *disjoints (n'ayant aucune variable commune) ne possède pas de variable* FAC*.*

Preuve 8.2. *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $\mathcal{M}_1 = \langle \mathcal{X}_1, \mathcal{C}_1 \rangle$ et $\mathcal{M}_2 = \langle \mathcal{X}_2, \mathcal{C}_2 \rangle$ deux* MUC *du $\mathcal{P}$ tel que $(\mathcal{X}_1 \cap \mathcal{X}_2) = \emptyset$. Raisonnons par l'absurde, supposons qu'il existe $X \in \mathcal{X}$ une variable* FAC *pour une certaine interprétation $\mathcal{I}$. Par définition des* MUC*, il existe $C_1 \in \mathcal{C}_1$ et $C_2 \in \mathcal{C}_2$ deux contraintes falsifiées par $\mathcal{I}$. Puisque X est une variable* FAC *pour $\mathcal{I}$, X appartient à toutes les contraintes falsifiées par $\mathcal{I}$. Donc $X \in C_1$ et $X \in C_2$, ce qui est absurde puisque $(\mathcal{X}_1 \cap \mathcal{X}_2) = \emptyset$.*

Finalement, comme le montre l'exemple suivant, nous pouvons noter que, dans le cadre d'instances satisfiables, il est tout de même possible de détecter des variables FAC. Ces variables peuvent jouer un rôle important dans la recherche d'une solution en mettant en exergue les parties difficiles du problème. De plus, lors d'un processus de recherche classique (MAC), l'affectation d'une variable peut conduire à obtenir un CSP inconsistant. La détection de variables FAC peut, dans ce contexte, aider à réfuter plus rapidement cette mauvaise décision.

Exemple 8.2. *Considérons le réseau de contraintes $\mathcal{P}$ représenté dans la figure 8.2 et l'interprétation complète $\mathcal{I} = \{(X_1 = 2), (X_2 = 1), (X_3 = 1), (X_4 = 1)\}$. Nous pouvons voir sur cette figure que, bien que le réseau de contraintes soit cohérent, la variable X_1 est une variable* FAC*. En effet, l'ensemble des contraintes falsifiée par l'interprétation $\mathcal{I}$ est $\{C_1, C_2, C_5\}$. Puisque $X_1 \in var(C_1) \cap var(C_2) \cap var(C_5)$ alors nous avons que X_1 est une variable* FAC*.*

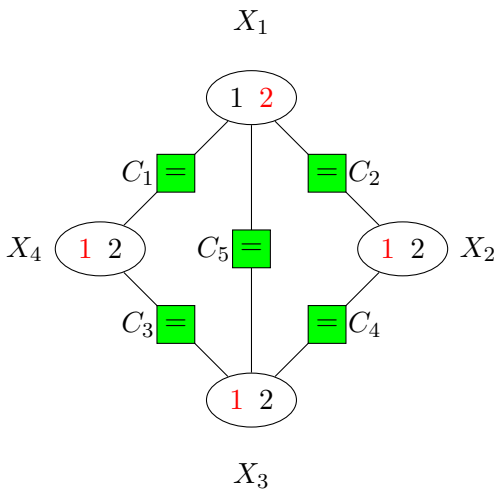


FIGURE 8.2 – Variable FAC d'un réseau de contraintes satisfiable obtenue à partir d'une interprétation complète.

À la vue des résultats obtenus par le solveur SATHYS, reportés dans le chapitre précédent, l'utilisation du concept de variables FAC et de la recherche locale dans le cadre d'une approche hybride semble être très prometteuse. L'approche FAC-SOLVER, décrite dans la section 8.3, tente d'implanter et de valider cette intuition de manière expérimentale en prenant en compte un large panel d'instances. Afin de produire une approche hybride efficace, nous proposons et étudions expérimentalement un solveur de recherche locale pour la résolution de CSP. Ce dernier, présenté dans la section suivante, s'appuie sur certaines metaheuristiques introduites dans le cadre SAT (voir section 2.2).

8.2 Solveur de recherche locale pour CSP

Dans cette section nous présentons nos contributions à la résolution pratique du problème CSP à l'aide de la recherche locale.

8.2.1 Algorithme de recherche locale

Comme nous l'avons déjà souligné dans la section 2.1.5, il est nécessaire, afin d'obtenir une méthode de recherche locale efficace, de se doter de structures de données évoluées permettant le développement d'algorithmes incrémentaux puissants. Pour notre solveur, nous avons choisi d'utiliser la méthode proposée par Galinier et Hao (1997). Cette dernière considère une structure de donnée γ qui, pour un réseau de contraintes $\langle \mathcal{X}, \mathcal{C} \rangle$, une interprétation complète $\mathcal{I}$ et un couple (X, v) tel que $X \in \mathcal{X}$ et $v \in dom(X)$, permet d'obtenir à tout moment le nombre $\gamma(X, v)$ de contraintes $C \in \mathcal{C}$ falsifiée par $\mathcal{I}$ tel que $X \in var(C)$.

L'algorithme 8.1 permet d'initialiser la structure de données γ (le prédicat `check`(C) permet de contrôler si C est satisfaite par l'interprétation actuelle des variables $var(C)$). Pour cela, étant donnée une interprétation complète $\mathcal{I}$, pour chaque valeur v de chaque variable X, $\gamma(X, v)$ est initialisée à zéro (ligne 4). Alors pour chaque contrainte C falsifiée par l'interprétation $\mathcal{I}_{\overline{X}} \cup \{(X = v)\}$ la valeur de $\gamma(X, v)$ est augmentée (lignes 5–7).

Algorithme 8.1 : initγ

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un CSP et $\mathcal{I}$ une interprétation complète
Résultat : La structure de données γ initialisée

```
1 Début
2     pour chaque X ∈ 𝒳 faire
3         pour chaque v ∈ dom(X) faire
4             γ(X, v) ← 0;
5             pour chaque C ∈ 𝒞 faire
6                 si (check(ℐ_X̄ ∪ {(X = v)}, C)) alors
7                     γ(X, v) ← γ(X, v) + 1;
8 Fin
```

Exemple 8.3. *La Figure 8.3 reporte, pour un réseau de contraintes $\mathcal{P}$ et une interprétation complète $\mathcal{I}$, les valeurs de la structure γ après l'appel de la fonction* `init`$_\gamma(\mathcal{P}, \mathcal{I})$.

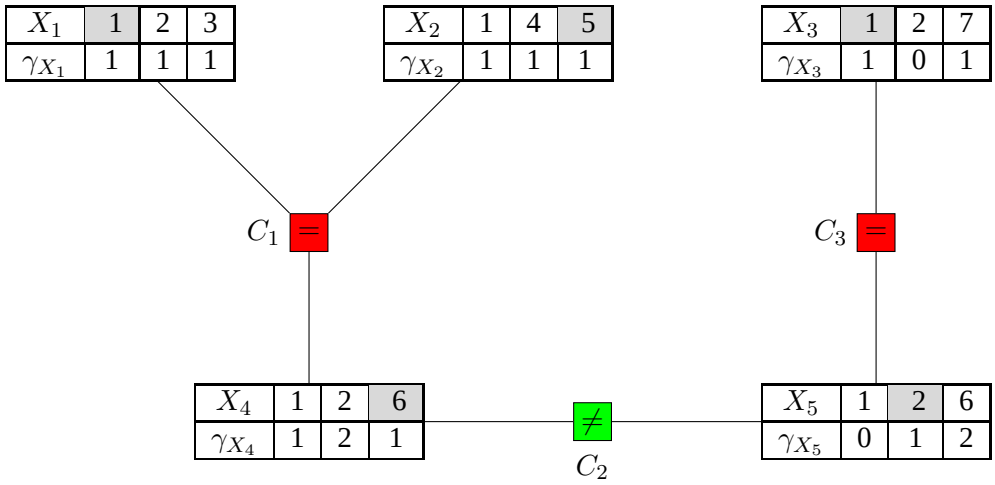


FIGURE 8.3 – Initialisation de la structure de données γ.

L'algorithme 8.2 permet d'effectuer la mise à jour de la structure γ dans le cas où un *switch* est effectué. Il prend en entrée un réseau de contraintes $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$, la variable X modifiée par le *switch*, $\mathcal{I}$ l'interprétation obtenu après le *switch* (l'interprétation courante) et $\mathcal{I}'$ l'interprétation précédente. Puisque la réparation de la variable X n'a d'impact que sur les contraintes qui l'implique, la mise à jour de γ se fait en considérant uniquement les contraintes liées à X. Plus précisément, pour chaque contrainte C telle que $X \in var(C)$, pour tout $v \in dom(Y)$ tel que $Y \in var(C)$ et Y différentes de X la valeur de $\gamma(Y, v)$ est mise à jour (lignes 2–9). Pour effectuer cela, il suffit pour chaque couple (Y, v) de vérifier si l'interprétation $\mathcal{I}_{\overline{Y}} \cup \{(Y = v)\}$ satisfait (respectivement falsifie) C et l'interprétation $\mathcal{I}'_{\overline{Y}} \cup \{(Y = v)\}$ falsifie (respectivement satisfait) C (ligne 5). En effet, la valeur de $\gamma(Y, v)$ change uniquement dans le cas où le *switch* de la valeur de X à un impact sur la satisfiabilité de la contrainte C. Dans ce cas, $\gamma(Y, v)$ est décrémentée si l'interprétation $\mathcal{I}_{\overline{Y}} \cup \{(Y = v)\}$ satisfait C (ligne 7), sinon $\gamma(Y, v)$ est incrémentée (ligne 9).

Algorithme 8.2 : updateγ

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un CSP, X une variable ainsi que $\mathcal{I}$ et $\mathcal{I}'$ deux interprétations complètes
Résultat : Met à jour la structures de données γ

```
1  Début
2      pour C ∈ 𝒞 | X ∈ var(C) faire
3          pour chaque Y ∈ var(C) tel que Y ≠ X faire
4              pour chaque v ∈ dom(Y) faire
5                  si (check(ℐ_Y ∪ {(Y = v)}, C) ≠ check(ℐ'_Y ∪ {(Y = v)}, C)) alors
6                      si (check(ℐ_Y ∪ {(Y = v)}, C)) alors
7                          γ(Y, v) ← γ(Y, v) − 1;
8                      sinon
9                          γ(Y, v) ← γ(Y, v) + 1;
10 Fin
```

L'algorithme 8.3 décrit la méthode de recherche locale que nous avons utilisé pour l'ensemble de nos expérimentations. Cette dernière, basée sur WSAT (voir algorithme 2.3), utilise la structure de données présentée précédemment afin d'évaluer le gain (en terme de contraintes violées) obtenu par le *switch* de la valeur d'une variable (ligne 10). Pour cela à chaque redémarrage, après avoir généré la nouvelle interprétation complète, la fonction `init`γ est appelée avec l'interprétation courante (ligne 4). Puis à chaque itération, la fonction `update`γ est appelé afin de prendre en compte la dernière réparation (ligne 15).

Algorithme 8.3 : WCSP

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un CSP, deux entiers $maxReparations$ et $maxEssais$
Résultat : `vrai` si $\mathcal{P}$ est satisfiable, `faux` si pas de modèle trouvé

```
1  Début
2      pour i de 0 à maxEssais faire
3          ℐ ← interprétation complète construite sur 𝒳;
4          initγ(𝒫, ℐ);
5          pour j de 0 à maxReparations faire
6              ℐ' ⟵ ℐ;
7              si ℐ ⊨ 𝒫 alors
8                  retourner vrai ;
9              Soit C ∈ false(𝒫, ℐ);
10             si (∃(X, v) ∈ var(C) × dom(X) tel que (γ(X, u) > γ(X, v)) et ℐ({X}) = {(X = u)})
               alors
11                 switch(ℐ, X, v);
12             sinon                                   /* minimum local */
13                 Choisir un couple (X, v) selon un critère d'échappement;
14                 switch(ℐ, X, v);
15             updateγ(P, X, ℐ, ℐ');
16     return faux ;
17 Fin
```

8.2.2 Mettre à jour γ plus rapidement

Dans la suite, nous proposons une structure de données dont le but est d'améliorer l'efficacité de la méthode `update`γ. Cette structure de données consiste pour chaque contrainte C du réseau à associer une matrice δ permettant de sauvegarder la satisfiabilité courante de la contrainte pour chaque couple (X, v) tel que $X \in var(C)$ et $v \in dom(X)$. De cette manière, lors de la mise à jour de la structure γ, il est possible d'économiser un appel à la fonction `check` (l'appel pour connaître la satisfiabilité de la contrainte pour l'interprétation précédente). Afin de considérer cette nouvelle structure, nous modifions légèrement les algorithmes 8.1 et 8.2.

L'algorithme 8.4 permet d'initialiser les structures données γ et δ. Ce dernier diffère de l'algorithme 8.1 uniquement par le fait qu'il affecte $\delta(C, X, v)$ à `vrai` si l'interprétation $\mathcal{I}_{\overline{X}} \cup \{X = v\}$ satisfait la contrainte C, sinon il affecte $\delta(C, X, v)$ à `faux` .

Algorithme 8.4 : init$\gamma\delta$

Données : $\langle \mathcal{X}, \mathcal{C} \rangle$ un CSP et $\mathcal{I}_c$ une interprétation complète
Résultat : Les structures de données γ et δ initialisées

```
1  Début
2      pour chaque X ∈ 𝒳 faire
3          pour chaque v ∈ dom(X) faire
4              γ(X, v) ← 0;
5              pour chaque C ∈ 𝒞 faire
6                  si check(I_X̄ ∪ {(X = v)}, C) = faux alors
7                      γ(X, v) ← γ(X, v) + 1;
8                      δ(X, v, C) ← false;
9                  sinon
10                     δ(X, v, C) ← true;
11 Fin
```

Exemple 8.4. *Considérons le réseau de contraintes $\mathcal{P}$ et l'interprétation complète $\mathcal{I}$ de l'exemple 8.3, la Figure 8.4 reporte les valeurs des structures γ et δ après l'appel de la fonction* `init`$\gamma\delta(\mathcal{P}, \mathcal{I})$.

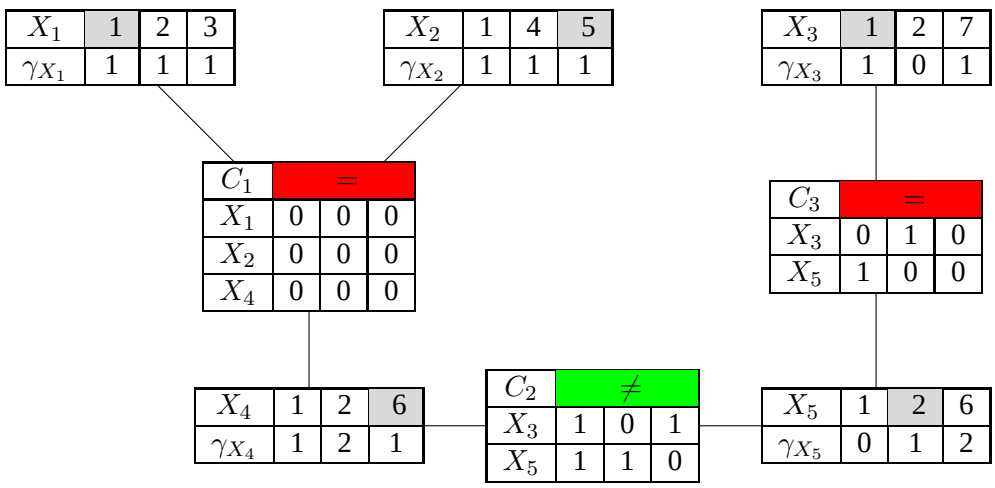


FIGURE 8.4 – Initialisation des structures de données γ et δ.

L'algorithme 8.5 permet de mettre à jour les structures γ et δ. Il ne diffère de l'algorithme 8.2 qu'au niveau de la mise à jour de la structure δ (lignes 8 et 11) et surtout au niveau du test de la ligne 5. En effet, contrairement à la méthode classique, il n'est pas nécessaire, puisque nous avons sauvegardé le résultat précédent dans la structure δ, de vérifier si l'interprétation $\mathcal{I}'_{\overline{Y}} \cup \{(Y = v)\}$ satisfait la contrainte C.

Algorithme 8.5 : update$\gamma\delta$

Données : $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un CSP, X une variable ainsi que $\mathcal{I}$ et $\mathcal{I}'$ deux interprétations complètes
Résultat : Met à jour les structures de données γ et δ

```
1  Début
2      pour C ∈ 𝒞|X ∈ var(C) faire
3          pour chaque Y ∈ var(C) tel que Y ≠ X faire
4              pour chaque v ∈ dom(Y) faire
5                  si (check(I'_Ȳ ∪ {(Y = v)}, C) ≠ δ(C, Y, v)) alors
6                      si (check(I'_Ȳ ∪ {(Y = v)}, C)) alors
7                          γ(Y, v) ← γ(Y, v) − 1;
8                          δ(C, Y, v) ← true;
9                      sinon
10                         γ(Y, v) ← γ(Y, v) + 1;
11                         δ(C, Y, v) ← false;
12 Fin
```

Afin de profiter de cette structure au sein de l'algorithme WCSP présenté précédemment, il suffit de remplacer `init`γ et `update`γ par `init`$\gamma\delta$ et `update`$\gamma\delta$ respectivement. La méthode résultant est nommée WCSP$_\delta$ dans la suite de ce chapitre.

8.2.3 Expérimentations

Dans cette section, nous étudions expérimentalement les performances de notre solveur de recherche locale vis-à-vis des critères d'échappements WALKSAT (Selman *et al.* 1994), NOVELTY et RNOVELTY (McAllester *et al.* 1997). Pour cette étude, nous avons considéré l'ensemble des instances de la compétitions CSP ayant eu lieu en 2008. Ces instances sont constituées d'instances binaires et n-aires, aléatoires et industrielles, satisfiables et insatisfiables. Ces instances ont été divisées en quatre catégories : 635 instances codant des contraintes binaires en extension (2-EXT), 696 instances codant des contraintes binaires en intention (2-INT), 704 instances codant des contraintes n-aires en extension (N-EXT) et 716 instances codant des contraintes n-aires en intention (N-INT).

Cette étude expérimentale est divisée en trois parties. Dans la première, nous étudions l'apport de la structure de données que nous avons proposé afin d'améliorer la vitesse de mise à jour de la structure γ. Ensuite, nous étudions l'apport de l'utilisation de la cohérence d'arc généralisée comme méthode de prétraitement. Pour terminer, nous comparons notre méthode par rapport à deux méthodes de recherche locale de l'état de l'art.

L'ensemble des tests reportés dans la suite de ce chapitre a été conduit sur un Intel Xeon 3.2 GHz (2 G RAM) sous Linux 2.6. Nous avons limité le temps à 1200 secondes et la mémoire à 900 Mbytes.

8.2.3.1 Implantation de la structure de données δ

Dans un premier temps, nous tentons d'évaluer l'apport de notre nouvelle structure de données. Pour cela, nous avons implémenté deux versions de l'approche WCSP : WCSP$_s$ la version n'utilisant pas notre

structure de données, et WCSP_δ la version utilisant la structure de donnée δ.

Le tableau 8.1 résume les résultats obtenus par WCSP_s et WCSP_δ pour les trois critères d'échappements (WALKSAT, NOVELTY et RNOVELTY). Comme nous pouvions le présumer, notre structure de données permet d'améliorer globalement les performances de notre méthode de recherche locale. Remarquons aussi que, puisque cette méthode est une approche de recherche locale, aucune instance insatisfiable ne peut être résolue.

	2-EXT			2-INT			N-EXT			N-INT		
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
WCSP_s(WALKSAT)	239	0	239	160	0	160	179	0	179	271	0	271
WCSP_δ(WALKSAT)	**248**	0	**248**	**167**	0	**167**	**181**	0	**181**	**283**	0	**283**
WCSP_s(NOVELTY)	308	0	308	159	0	159	222	0	222	338	0	338
WCSP_δ(NOVELTY)	**315**	0	**315**	**165**	0	**165**	**223**	0	**223**	**350**	0	**350**
WCSP_s(RNOVELTY)	319	0	319	161	0	161	208	0	208	321	0	321
WCSP_δ(RNOVELTY)	**325**	0	**325**	**173**	0	**173**	**209**	0	**209**	**325**	0	**325**

TABLE 8.1 – Étude de l'apport de la structure de données δ pour l'efficacité de la méthode de recherche locale. Pour chaque type de problème et pour chaque solveur, nous reportons le nombre d'instances satisfaisables, insatisfaisables et totales résolues.

Afin d'estimer l'impact de notre structure de données sur la vitesse de résolution de notre méthode de recherche locale, nous reportons dans les trois nuages de points suivants les comparaisons faites entre WCSP_s(WALKSAT) et WCSP_δ(WALKSAT) (figure 8.5), entre WCSP_s(NOVELTY) et WCSP_δ(NOVELTY) (figure 8.6) et finalement entre WCSP_s(RNOVELTY) et WCSP_δ(RNOVELTY) (figure 8.6). Sur ces dernières, nous pouvons voir que l'ajout de notre structure de données réduit significativement les temps de résolution.

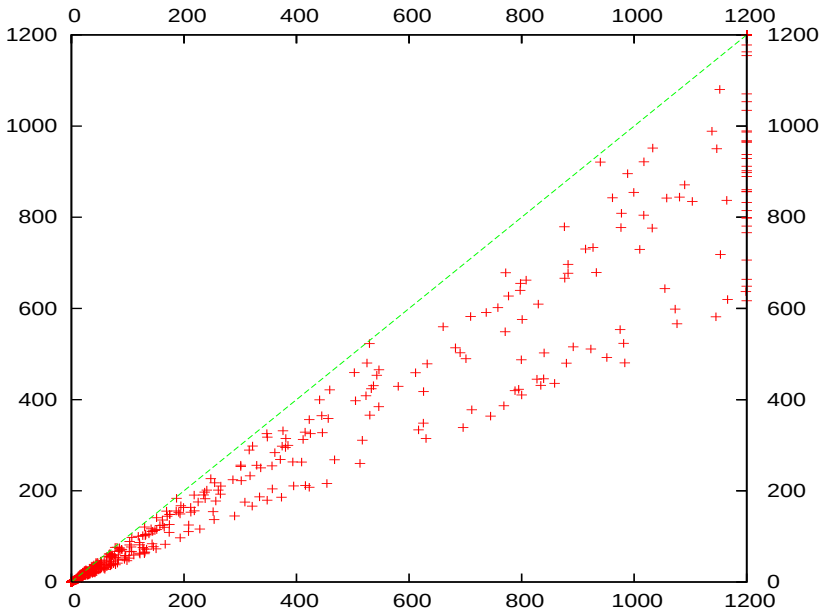


FIGURE 8.5 – Comparaison des temps de résolution de WCSP_s(WALKSAT) et WCSP_δ(WALKSAT). Un point (temps WCSP_s(WALKSAT), temps WCSP_δ(WALKSAT)) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

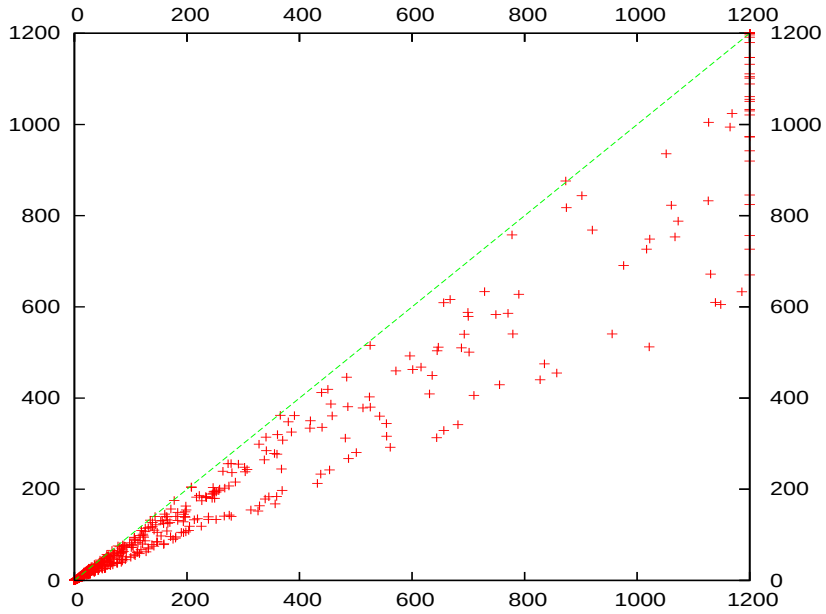


FIGURE 8.6 – Comparaison des temps de résolution de $\text{WCSP}_s(\text{NOVELTY})$ et $\text{WCSP}_\delta(\text{NOVELTY})$. Un point (temps $\text{WCSP}_s(\text{NOVELTY})$, temps $\text{WCSP}_\delta(\text{NOVELTY})$) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

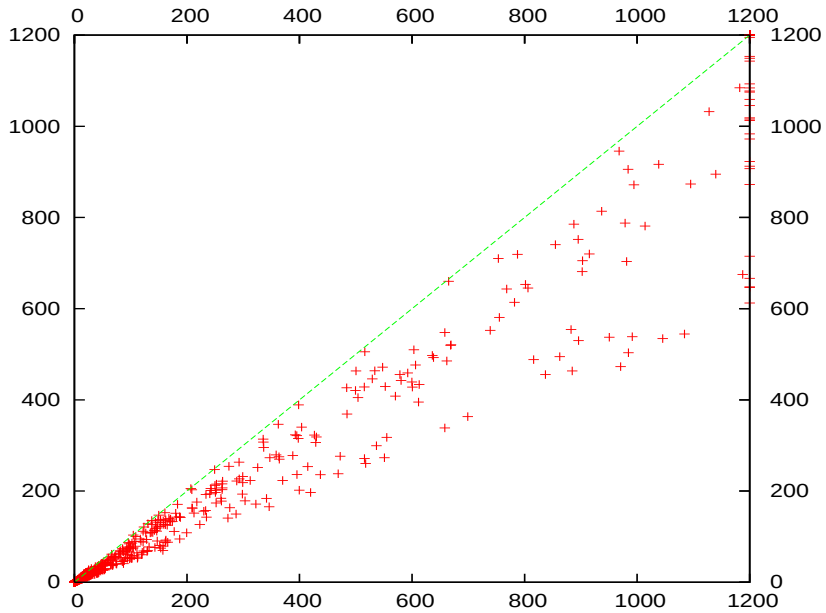


FIGURE 8.7 – Comparaison des temps de résolution de $\text{WCSP}_s(\text{RNOVELTY})$ et $\text{WCSP}_\delta(\text{RNOVELTY})$. Un point (temps $\text{WCSP}_s(\text{RNOVELTY})$, temps $\text{WCSP}_\delta(\text{RNOVELTY})$) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

8.2.3.2 GAC en prétraitement

Dans cette étude, nous tentons d'évaluer l'apport pratique et théorique de l'application de la cohérence d'arc généralisée en tant que méthode de prétraitement.

Le tableau 8.2 récapitule les résultats obtenus par notre méthode de recherche locale dans le cas où la cohérence d'arc généralisée est effectuée en prétraitement ($\text{WCSP}^*_\delta(\text{WALKSAT})$, $\text{WCSP}^*_\delta(\text{NOVELTY})$, $\text{WCSP}^*_\delta(\text{RNOVELTY})$) et dans celui où aucun processus de prétraitement n'est réalisé ($\text{WCSP}_\delta(\text{WALKSAT})$, $\text{WCSP}_\delta(\text{NOVELTY})$, $\text{WCSP}_\delta(\text{RNOVELTY})$). Nous pouvons voir dans ce tableau que l'application de la cohérence d'arc généralisée en prétraitement permet systématiquement d'améliorer les performances de la méthode.

	2-EXT			2-INT			N-EXT			N-INT		
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
$\text{WCSP}_\delta(\text{WALKSAT})$	248	0	248	167	0	167	181	0	181	283	0	283
$\text{WCSP}^*_\delta(\text{WALKSAT})$	**270**	0	**270**	**208**	**82**	**290**	**183**	**30**	**213**	**295**	**8**	**303**
$\text{WCSP}_\delta(\text{NOVELTY})$	315	0	315	165	0	165	223	0	223	350	0	350
$\text{WCSP}^*_\delta(\text{NOVELTY})$	**348**	0	**348**	**197**	**82**	**279**	**230**	**30**	**260**	**357**	**8**	**365**
$\text{WCSP}_\delta(\text{RNOVELTY})$	325	0	325	173	0	173	209	0	209	325	0	325
$\text{WCSP}^*_\delta(\text{RNOVELTY})$	**358**	0	**358**	**200**	**82**	**282**	**215**	**30**	**245**	**330**	**8**	**338**

TABLE 8.2 – Étude de l'apport de l'application de l'arc consistance généralisée en tant que prétraitement d'une méthode de recherche locale. Pour chaque type de problème et pour chaque solveur, nous reportons le nombre d'instances satisfaisables, insatisfaisables et totales résolues.

L'apport de la cohérence d'arc généralisée en prétraitement d'une méthode de recherche locale est double. D'une part, elle permet de réduire la taille du domaine de certaines variables et par conséquent de montrer l'insatisfiabilité du réseau de contraintes ou de réduire la difficulté du problème restant à résoudre. D'autre part, l'utilisation d'un tel filtrage permet, comme l'énonce la propriété 8.3, de s'assurer qu'il est toujours possible de trouver une interprétation voisine permettant de satisfaire une contrainte falsifiée par l'interprétation courante.

Propriété 8.3. *Soient $\mathcal{P} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes binaires, $\mathcal{I}$ une interprétation complète conflictuelle des variables de $\mathcal{X}$ et $C \in \mathcal{C}$ une contrainte falsifiée par $\mathcal{I}$. Si $\mathcal{P}$ satisfait la cohérence d'arc alors $\exists \mathcal{I}' \in \mathcal{N}(\mathcal{I}, C)$ tel que $\mathcal{I}'$ satisfait C.*

Preuve 8.3. *Supposons, sans perte de généralité, que $var(C) = \{X, Y\}$ et que $\mathcal{I}_{\overline{X}} = \{X = u\}$. Puisque $\mathcal{P}$ satisfait la cohérence d'arc, $\exists v \in dom(Y)$ tel que (u, v) est un tuple autorisé de C. Par conséquent, l'interprétation $\mathcal{I}' = \mathcal{I}_{\overline{Y}} \cup \{Y = v\}$ satisfait la contrainte C. Étant donné que $\mathcal{I}$ et $\mathcal{I}'$ ne diffèrent que sur la valeur de la variable Y, $\mathcal{I}' \in \mathcal{N}(\mathcal{I}, C)$.*

Comme nous avons pu le voir sur le tableau 8.2, cette propriété permet d'améliorer sensiblement les performances de notre méthode sur les réseaux de contraintes binaires. Malheureusement, elle ne peut être étendue aux réseaux de contraintes $n - aire$. En effet, comme le montre la figure 8.8, il n'est pas toujours possible, pour une contrainte satisfaisant la cohérence d'arc généralisée et une interprétation complète $\mathcal{I}$, de trouver une interprétation $\mathcal{I}' \in \mathcal{N}(\mathcal{I}, C)$ qui satisfait la contrainte C. Ceci permet en partie d'expliquer les raisons qui font que l'application de la cohérence d'arc généralisée ne permet pas d'accroître de manière plus notable les performances de notre solveur de recherche locale sur les instances $n - aires$.

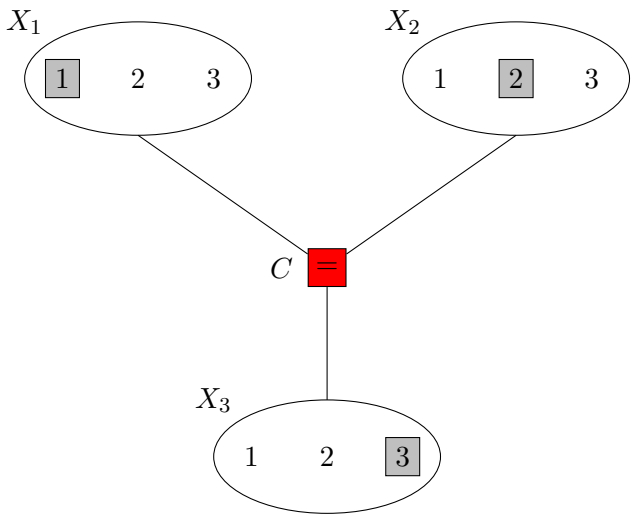


FIGURE 8.8 – Contrainte non satisfaite quelle que soit l'interprétation voisine considérée.

8.2.3.3 Étude comparative

Nous reportons dans le tableau 8.3 les résultats obtenus par notre approche de recherche locale vis-à-vis de deux méthodes de recherche locale implémentées dans le solveur CSP4J proposée par Vion (2007) :

– CSP4J(TABOU), recherche locale basée sur l'approche TABOU (Galinier et Hao 1997) ;
– CSP4J(WMC), recherche locale basée sur l'heuristique *Min-Conflit* (Minton *et al.* 1992).

L'ensemble des méthodes utilisées effectue GAC en prétraitement.

Nous pouvons clairement voir que, quel que soit le critère d'échappement utilisé, notre méthode résout globalement plus d'instances que les méthodes CSP4J(TABOU) et CSP4J(WMC). Plus précisément, nous pouvons voir que, hormis sur les instances insatisfiables en intention $n - aire$ (ceci n'est pas imputable à notre méthode de recherche locale mais plutôt à l'algorithme de filtrage utilisé en prétraitement), notre méthode résout systématiquement plus d'instances.

	2-EXT			2-INT			N-EXT			N-INT		
Méthode	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
WCSP$^*_\delta$(WALKSAT)	270	0	270	**208**	**82**	**290**	183	30	213	295	8	303
WCSP$^*_\delta$(NOVELTY)	349	0	349	197	**82**	279	**230**	30	**260**	**358**	8	**366**
WCSP$^*_\delta$(RNOVELTY)	**358**	0	**358**	200	**82**	282	215	30	245	330	8	338
CSP4J(TABOU)	342	0	342	143	35	178	174	30	204	170	**21**	191
CSP4J(WMC)	114	0	114	189	33	222	100	30	130	151	**21**	172

TABLE 8.3 – Comparaison du solveur WCSP vis-à-vis de solveurs de l'état de l'art. Pour chaque type de problème et pour chaque solveur, nous reportons le nombre d'instances satisfaisables, insatisfaisables et totales résolues.

8.2.4 Synthèse

Dans cette section, nous avons défini une nouvelle méthode de recherche locale pour la résolution pratique du problème CSP. Comme nous avons pu le voir lors de la dernière expérimentation, cette

méthode est très compétitive. Dans la suite de ce chapitre, nous utilisons ce solveur comme base de notre méthode FAC-SOLVER.

8.3 L'approche FAC-SOLVER

Le solveur FAC-SOLVER intègre trois types d'approches en son sein : une recherche locale, un solveur de type MAC et un solveur hybride qui est une combinaison d'un solveur de recherche locale et du processus de filtrage GAC (voir section 3.2.3.1). Ces composantes interagissent ensemble de plusieurs manières (pondération, détection de FAC...) et partagent l'ensemble des informations obtenues au cours de recherche (variables FAC, valeurs supprimées au niveau 0). L'automate décrit dans la Figure 8.9 donne le schéma général de notre approche. Pour commencer, le processus appelle la recherche locale sur le problème initial. Dans le cas où la recherche locale échoue à trouver une solution dans le temps qui lui a été imparti (contrôlé par $slsProgress$), la partie hybride (RL+GAC) prend la main. Cette partie consiste à rendre taboues les variables problématiques pour la recherche locale en les fixant à l'aide d'un processus complet (affectation et filtrage). Enfin, notre approche passe la main à la partie MAC lorsqu'elle n'a pas été capable de fournir un résultat avant d'avoir atteint un nombre de conflits fixé à l'avance. Cette dernière partie est simplement un solveur MAC classique avec une heuristique de choix de variables basée sur la notion de variables FAC présentée précédemment. Comme pour la partie RL+GAC, la partie MAC redonne finalement la main à la recherche locale (nouveau cycle) lorsqu'un certain nombre de conflits a été atteint.

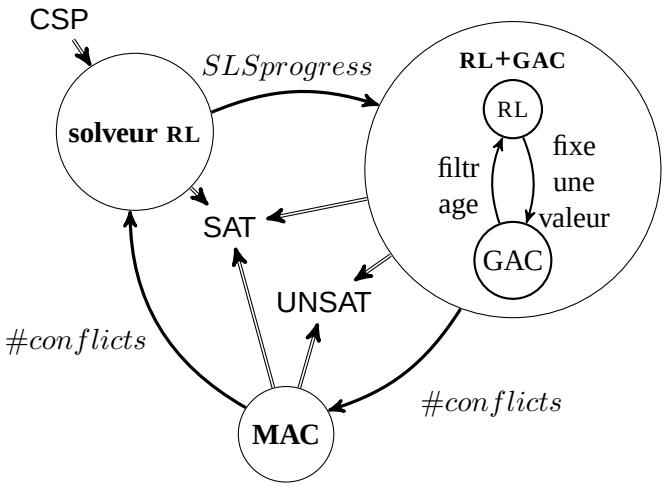


FIGURE 8.9 – Interactions entre les différentes composantes de FAC-SOLVER.

8.3.1 FAC-SOLVER

L'algorithme 8.6 décrit le solveur FAC-SOLVER. En premier lieu, l'ensemble des variables utiles aux différentes composantes de notre solveur est initialisé : le nombre de conflits contrôlant le redémarrage associé aux parties RL+GAC et MAC du solveur est initialisé à 10, l'ensemble contenant les variables FAC détectées est affecté à vide et une interprétation complète des variables est générée de manière aléatoire (ligne 2–6). Cette interprétation sera utilisée par la recherche locale. Ensuite, un appel à la procédure GAC est effectué afin d'assurer l'arc-consistance ou de montrer que le problème est directement incohérent (lignes 4–5). Une fois ce préambule terminé, tant qu'une solution n'a pas été trouvée ou que l'absence

de solution n'a pas été établie, le solveur effectue de manière séquentielle les trois composantes RL, RL+GAC et MAC.

Algorithme 8.6 : FAC-solver

Données : Un CSP $P = (\mathcal{X}, \mathcal{C})$
Résultat : `vrai` si le CSP est satisfiable, `faux` sinon

```
1  Début
2      result ⟵ unknown ;
3      maxConf ⟵ 10 ;
4      S_e ⟵ ∅ ;                                   //ensemble de var FAC
5      I ⟵ une interprétation complète de X choisie aléatoirement ;
6      GAC();
7      si ∃X ∈ X s.t. dom(X) = ∅ alors
8          retourner false ;
9      tant que (result = unkown) faire
10         initialiser la variable slsProgress;
11         RL(⟨X, C⟩) ;
12         si (result ≠ unkown) alors
13             retourner true
14         RL+GAC(⟨X, C⟩) ;
15         si (result ≠ unkown) alors
16             retourner result
17         MAC(⟨X, C⟩) ;
18         si (result ≠ unkown) alors
19             retourner result
20         maxConf ⟵ maxConf × 1.5 ;
21         Backjump(0);                    // redémarrage/nouveau cycle
22 Fin
```

Avant de détailler chacune des composantes, nous décrivons, de manière intuitive, leurs interactions et apports respectifs. Pour la RL, en plus d'essayer de trouver une interprétation satisfaisant l'ensemble des contraintes du CSP, cette composante permet de détecter et collecter des variables FAC. Concernant la partie RL+GAC, elle permet d'aider la recherche locale (en vérifiant que certaines de ces affectations sont bien arc-cohérentes) et permet de pondérer les contraintes les plus souvent falsifiées durant la RL (choix des variables limité au *scope* des contraintes falsifiées). Quant à la composante MAC, elle tente de résoudre le problème en intégrant l'ensemble des informations recueillies par les deux autres composantes (FAC et ajustement du poids des contraintes).

Il est important de noter que, puisque $maxConf$ est augmenté de manière géométrique (ligne 20), la composante MAC donnera un résultat lorsque la borne ($maxConf$) deviendra plus grande que le nombre de conflits nécessaires pour résoudre le CSP. Ainsi, la complétude de notre méthode est garantie.

8.3.2 La composante recherche locale : RL

La procédure 8.7 présente de manière succincte la composante représentant la partie recherche locale de notre solveur. Cette procédure est basée sur l'approche WCSP décrite dans la section précédente. Elle utilise l'heuristique NOVELTY comme critère d'échappement (McAllester *et al.* 1997) aux minima locaux

(voir section 2.2). Cette méthode identifie également des variables FAC à chaque minimum local. La variable contrôlant l'avancement de la recherche est $slsProgress$, elle est augmentée dans deux situations : quand une nouvelle valeur de *maxCSP* (nombre de contraintes falsifiées minimum jamais trouvé) est atteinte (ligne 16) et lorsqu'une nouvelle variable FAC est découverte (ligne 6). Elle est décrémentée dans le cas où un minimum local est atteint et qu'aucune variable FAC n'a été trouvée (ligne 8). Cette variable est initialisée à 10000 si le CSP est binaire et 1000 sinon. Cette manière d'estimer le progrès de la recherche locale est inspirée des travaux introduit par Hoos (2002) concernant le réglage des différents paramètres utilisés en recherche locale dans le cadre de SAT (voir section 2.3). Ce paramètre est fondamental pour l'efficacité de l'hybridation et permet d'évaluer dynamiquement la progression de la méthode de recherche locale dans son exploration stochastique de l'espace de recherche. Lorsque la recherche locale échoue à trouver une solution du CSP et qu'elle se trouve « engluée », le test $slsProgress < 0$ (ligne 10) permet de passer la main à l'algorithme hybride qui va utiliser cette situation d'échec comme point de départ.

Procédure 8.7 : RL($\langle \mathcal{X}, \mathcal{C} \rangle$)

```
1  Début
2      tant que ∃C ∈ 𝒞 telle que C est falsifiée par ℐ faire
3          si minimum local est atteint alors
4              si ∃ variable FAC alors
5                  Ajouter une nouvelle variable FAC à 𝒮_e;
6                  slsProgress ⟵ slsProgress + 1000 ;
7              sinon
8                  slsProgress ⟵ slsProgress − 1 ;
9              si slsProgress < 0 alors
10                 retourner ;
11             sinon
12                 Changer la valeur dans ℐ d'une variable de 𝒳 en fonction du critère d'échappement
                   novelty;
13         sinon
14             Changer la valeur dans ℐ d'une variable de 𝒳 tel que le nombre de contraintes falsifiées
               diminue;
15         si un nouveau maxCSP est obtenu alors
16             slsProgress ⟵ slsProgress + 1000 ;
17     result ⟵ true ;
18 Fin
```

8.3.3 La composante hybrid : RL+GAC

La procédure 8.8 décrit la partie RL+GAC de notre solveur. Étant donnée la dernière interprétation $\mathcal{I}$ explorée par la recherche locale, cette interprétation ne permettait plus à la recherche locale de progresser (suivant notre critère) et va donc être utilisée par la composante RL+GAC pour fixer une valeur à une variable. Tant qu'un certain nombre de conflits n'a pas été atteint (ligne 4), la procédure sélectionne une variable X appartenant à une contrainte falsifiée par $\mathcal{I}$ (en utilisant l'heuristique $dom/wdeg$ Boussemart *et al.* (2004)) et tente de fixer une de ses valeurs à l'aide de la procédure FIX (lignes 5–7). La valeur est choisie de telle sorte que ce soit la valeur selon laquelle X est assignée dans $\mathcal{I}$. Une fois cette variable fixée le problème peut être montré incohérent ($result = false$) et la procédure se termine. Sinon, un

appel à la procédure RL est effectué. Mais les variables fixées durant cette étape ne pourront pas être modifiées par la recherche locale.

Procédure 8.8 : RL+GAC($\langle \mathcal{X}, \mathcal{C} \rangle$)

```
1  Début
2      level ⟵ 0;
3      #conf ⟵ 0 ;
4      tant que (#conf < maxConf) faire
5          X ⟵ choisir une variable X ∈ false(𝒳, 𝒞, ℐ) à l'aide de l'heuristique dom/wdeg;
6          v ⟵ the value of X in ℐ ;
7          FIX(⟨𝒳, 𝒞⟩, X, v) ;
8          si (result = false) alors
9              retourner ;
10         RL(⟨𝒳, 𝒞⟩) ;
11 Fin
```

La procédure FIX, qui peut être vue comme une partie d'un solveur MAC, permet de gérer la partie affectation et propagation. Elle consiste à assigner la valeur v à la variable X (ligne 2) et à effectuer l'arc-cohérence généralisée sur le problème résultant (ligne 4). Ensuite, tant que le CSP est conflictuel (c'est -à-dire qu'il existe une variable dont le domaine est vide) la procédure effectue un *backtrack* permettant de rétablir l'état de chaque variable à l'état $level - 1$, décrémente le niveau, réfute la valeur v_{level} de la variable X_{level} affectée au niveau $level$ et effectue de nouveau GAC (lignes 9–13). Un niveau correspond donc au nombre de valeurs fixées heuristiquement par RL+GAC ou MAC.

Procédure 8.9 : FIX($\langle \mathcal{X}, \mathcal{C} \rangle, X, v$)

```
1  Début
2      ⟨𝒳, 𝒞⟩ ⟵ ⟨𝒳, 𝒞⟩|_{X=v};
3      level ⟵ level + 1;
4      GAC() ;
5      tant que ∃X' ∈ 𝒳 tel que dom(X') = ∅ faire
6          si level = 0 alors
7              result ⟵ false ;
8              retourner ;
9          ⟨𝒳, 𝒞⟩ ⟵ Backtrack() ;
10         level ⟵ level − 1;
11         #conf ⟵ #conf + 1 ;
12         ⟨𝒳, 𝒞⟩ ⟵ ⟨𝒳, 𝒞⟩|_{X_level ≠ v_level};
13         GAC() ;
14 Fin
```

8.3.4 La composante MAC

La procédure suivante décrit la procédure MAC.

Procédure 8.10 : MAC($\langle \mathcal{X}, \mathcal{C} \rangle$)

```
1  Début
2      Backjump(0) ;                                   // redémarrage
3      level ⟵ 0;
4      #conf ⟵ 0 ;
5      tant que (#conf < maxConf) faire
6          si X = ∅ alors
7              result ⟵ true ;
8              retourner ;
9          si (#conf = 0) and (∃X ∈ S_e ∩ X) alors
10             X ⟵ choisir une variable dans S_e ;
11         sinon
12             X ⟵ choisir une variable selon dom/wdeg ;
13         v ⟵ choisir aléatoirement une valeur dans dom(X) ;
14         FIX(⟨X, C⟩, X, v) ;
15         si (result = false) alors
16             retourner
17 Fin
```

La composante MAC commence avec le CSP initial modulo les valeurs filtrées au niveau 0 durant les appels précédents à la procédure FIX faits durant RL+GAC. Cette composante diffère des solveurs MAC classiques par l'heuristique de choix de variables. En effet, tant qu'un conflit n'a pas été atteint le prochain point de choix est choisi parmi l'ensemble des variables FAC. Ensuite, l'heuristique de choix de variables devient $dom/wdeg$ (ligne 12) afin de ne pas disperser la recherche sur différentes zones d'incohérences. Cet algorithme n'effectue pas nécessairement une recherche complète puisque si le nombre de conflits $\#conf$ devient supérieur à $\#conf$ avant d'avoir résolu le problème, alors le processus s'arrête et un nouveau cycle est réalisé (RL $\rightarrow$ RL+GAC $\rightarrow$ MAC $\hookleftarrow$). Afin de garantir la complétude de la méthode, le nombre de conflits autorisés est augmenté avant chaque nouveau cycle (ligne 15 de l'algorithme 8.6).

8.4 Résultats expérimentaux

Afin d'étudier séparément chacune des composantes, nous avons testé quatre méthodes sur l'ensemble des instances de la compétitions CSP 2008 : NOVELTY qui est notre implémentation de la recherche locale (WCSP$^*_\delta$(NOVELTY)), RL+GAC, MAC et notre approche FAC-SOLVER.

Le tableau 8.4 récapitule les résultats, en terme de nombres d'instances SAT et UNSAT résolues par les différentes approches. Pour chaque catégorie d'instances le total est reporté. Sur ces lignes de totaux, les résultats du meilleur solveur sont grisés. Le principal constat que nous pouvons tirer de ce tableau est que notre approche FAC-SOLVER résout globalement le plus d'instances (que ce soit SAT ou UNSAT), et qu'elle est la meilleure sur trois des quatre catégories d'instances considérées. Pour le dernier type d'instances (binaires en extension), nous pouvons voir que les meilleurs solveurs sont différents pour chaque catégorie (SAT, UNS(AT) et TOT(AL)) et que le nombre d'instances résolues par FAC-SOLVER est toujours très proche du meilleur résultat obtenu.

		NOVELTY			RL+GAC			MAC			FAC-SOLVER		
		SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
2-EXT	ACAD	7	0	7	7	2	9	7	2	9	7	2	9
	PATT	106	0	106	100	38	138	83	38	121	99	39	138
	QRND	24	0	24	24	51	75	24	51	75	24	51	75
	RAND	206	0	206	197	105	302	194	110	304	193	106	299
	REAL	6	0	6	7	0	7	7	0	7	7	0	7
	TOTAL	349	0	349	335	196	531	315	201	516	330	198	528
2-INT	ACAD	38	7	45	37	40	77	37	40	77	38	40	78
	BOOL	0	1	1	0	1	1	0	1	1	0	1	1
	PATT	112	0	112	150	60	210	146	62	208	152	62	214
	REAL	47	74	121	74	102	176	75	103	178	75	103	178
	TOTAL	197	82	279	261	203	464	258	206	464	265	206	471
N-EXT	BOOL	70	1	71	74	75	149	74	70	144	74	74	148
	PATT	6	0	6	30	0	30	29	0	29	30	0	30
	QRND	43	0	43	40	40	80	33	40	73	45	40	85
	RAND	70	0	70	68	32	100	72	34	106	70	34	104
	REAL	41	29	70	45	114	159	47	115	162	47	115	162
	TOTAL	230	30	260	257	261	518	255	259	514	266	263	529
N-INT	ACAD	40	0	40	39	23	62	36	23	59	40	23	63
	BOOL	145	1	146	156	12	168	146	12	158	162	13	175
	PATT	88	5	93	103	19	122	95	20	115	102	18	120
	REAL	85	2	87	152	3	155	150	3	153	152	3	155
	TOTAL	358	8	366	450	57	507	427	58	485	456	57	513
	TOTAL	**1134**	**113**	**1247**	**1293**	**717**	**2010**	**1255**	**724**	**1979**	**1317**	**724**	**2041**

TABLE 8.4 – Comparaison du solveur FAC-SOLVER vis-à-vis des différentes parties qui le composent. Pour chaque catégorie de problème et pour chaque solveur, nous reportons le nombre d'instances satisfiables, insatisfaisables et totales résolues.

Afin d'étudier plus finement notre solveur, nous reportons sur les cinq nuages de points qui suivent les résultats obtenus par chacune des composantes de FAC-SOLVER sur l'ensemble des instances de la compétition CSP 2008 : pour chaque couple possible, le nuage de points résultant permet de corréler le temps mis par chacune des deux méthodes pour résoudre une instance donnée (comparaison entre FAC-SOLVER et NOVELTY (figure 8.10), FAC-SOLVER et RL+GAC (figures 8.11 et 8.13) et FAC-SOLVER et MAC (figures 8.12 et 8.14)). Pour toutes les figures, le solveur FAC-SOLVER est représenté sur l'axe des abscisses tandis que l'approche comparée est reportée sur l'axe des ordonnées. Les résultats, reportés en secondes, sont visualisés à l'aide d'une échelle logarithmique. Ces comparaisons ont été réalisées sur les instances SAT et UNSAT à l'exception de la comparaison avec $\text{WCSP}_\delta + \text{NOVELTY}$ où seules les instances SAT ont été considérées, la méthode de recherche locale étant incapable de prouver l'incohérence d'un CSP. Les principales informations pouvant être extraites sont les suivantes :

- il y a plus d'instances situées sur la ligne Y=1200 que sur la ligne X=1200. Ceci montre, comme cela est reporté dans le tableau 8.4, que FAC-SOLVER résout plus d'instances que les autres méthodes ;
- à l'exception du solveur MAC sur les instances UNSAT, on peut voir qu'il y a beaucoup de points au dessus de la diagonale, montrant ainsi que FAC-SOLVER est généralement plus efficace que les autres méthodes. De plus, pour les instances UNSAT, nous pouvons voir que la différence entre les différentes paires de solveurs est plus faible que sur les instances SAT (les points sont moins dispersés et plus proches de la diagonale). En ce qui concerne la comparaison de notre approche avec RL+GAC, nous pouvons voir que FAC-SOLVER est globalement plus efficace. Comparativement à l'approche MAC, notre méthode est meilleure sur les instances SAT (apport de la recherche locale) et légèrement moins bonne (en temps) sur les instances UNSAT. Ce décalage de temps sur les instances UNSAT s'explique par le temps utilisé par la recherche locale pour collecter des informations.

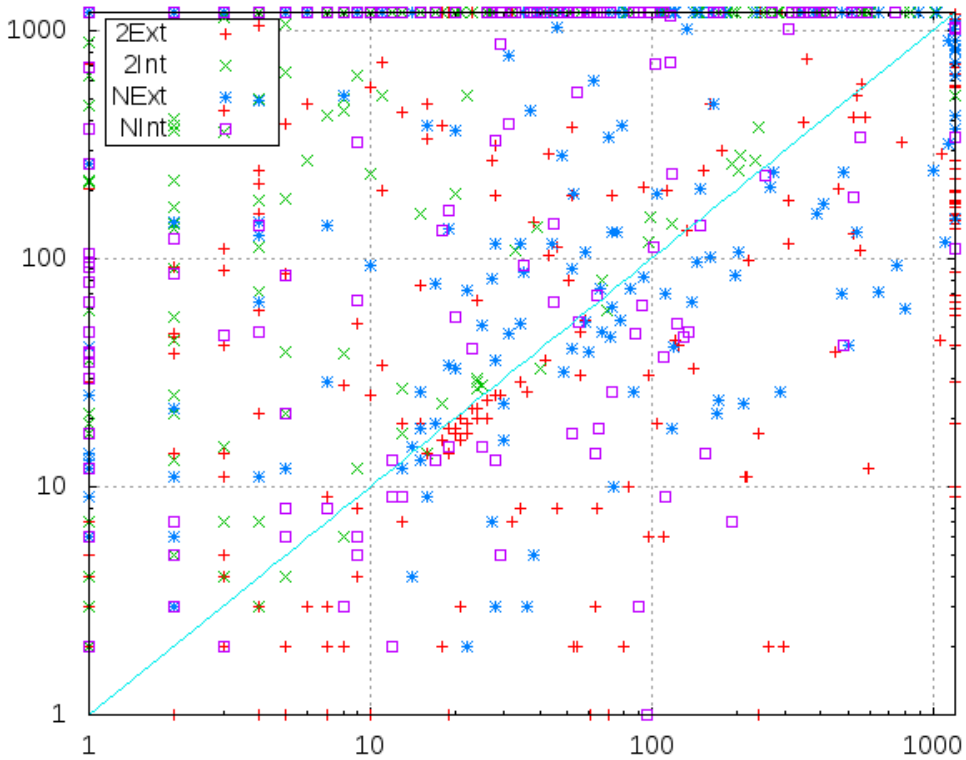


FIGURE 8.10 – Comparaison de FAC-SOLVER et NOVELTY sur les instances satisfiables. Chaque point (FAC-SOLVER, NOVELTY) reporté correspond au résultat obtenu par chacun des solveurs sur une instance.

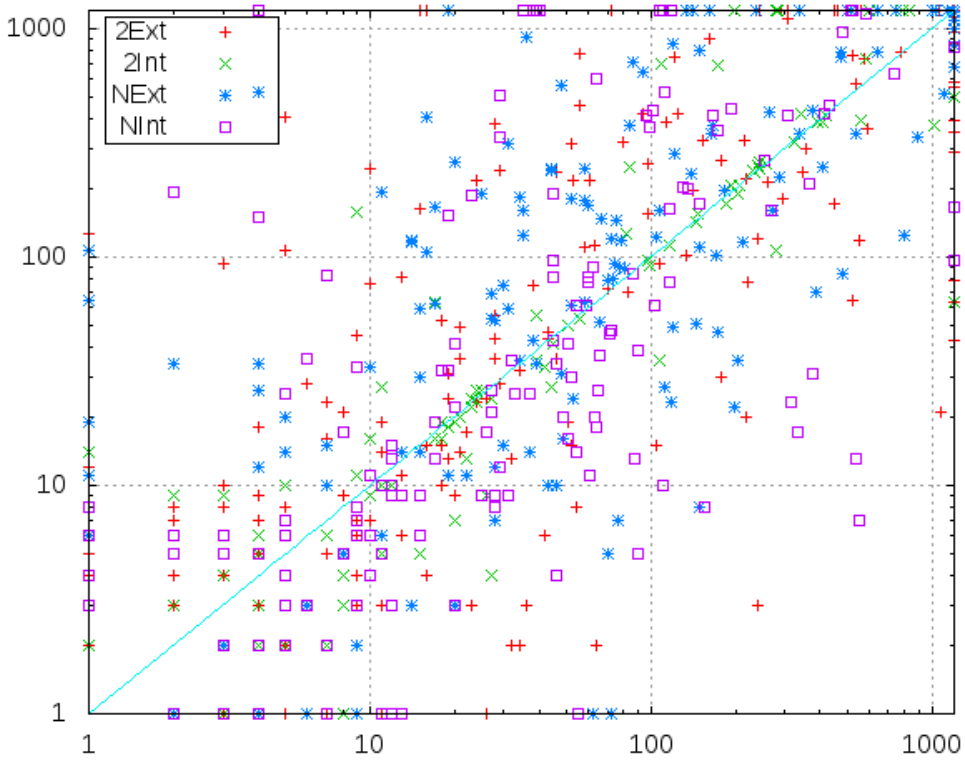


FIGURE 8.11 – Comparaison de FAC-SOLVER et RL+GAC sur les instances satisfiables. Chaque point (FAC-SOLVER, RL+GAC) reporté correspond au résultat obtenu par chacun des solveurs sur une instance.

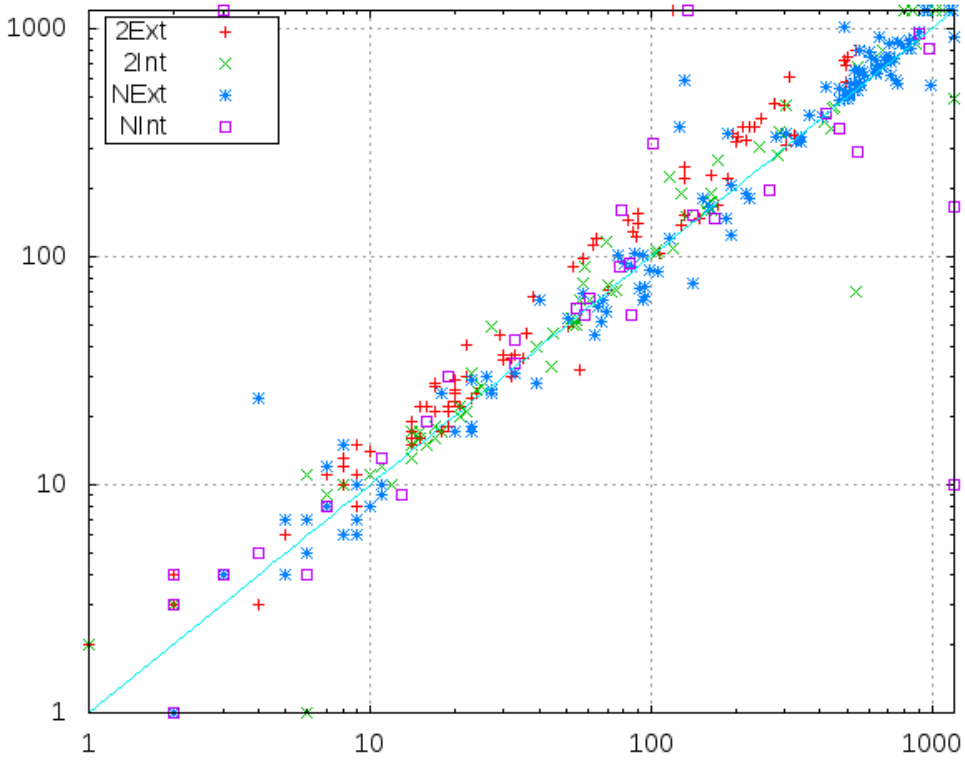


FIGURE 8.12 – Comparaison de FAC-SOLVER et MAC sur les instances satisfiables. Chaque point (FAC-SOLVER, MAC) reporté correspond au résultat obtenu par chacun des solveurs sur une instance.

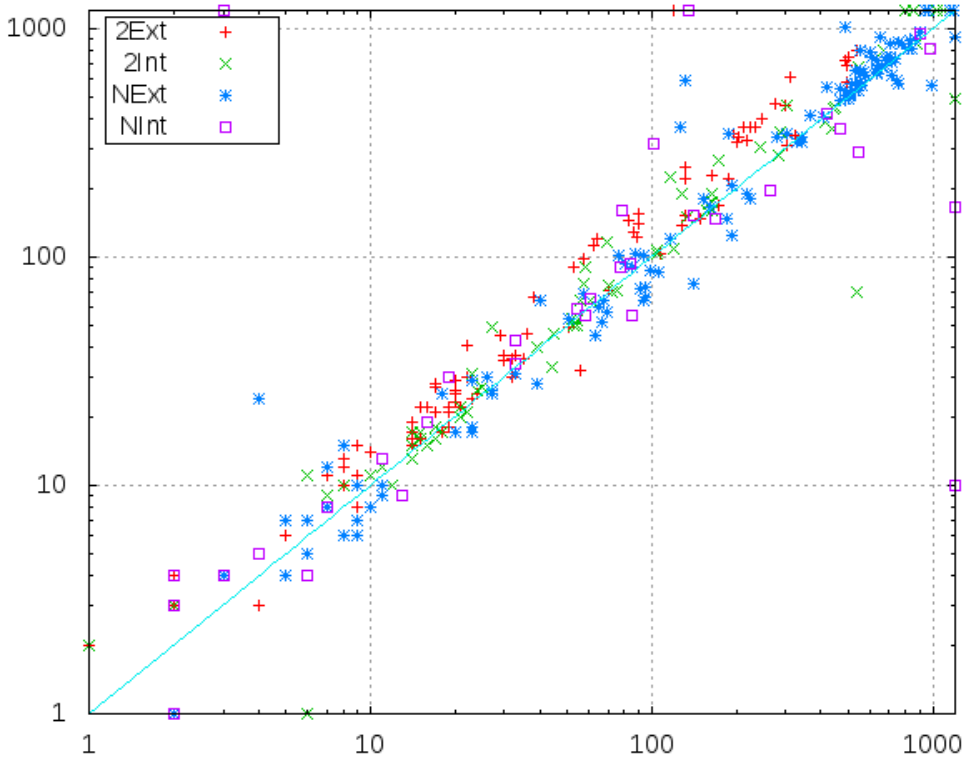


FIGURE 8.13 – Comparaison de FAC-SOLVER et RL+GAC sur les instances insatisfiables. Chaque point (FAC-SOLVER, RL+GAC) reporté correspond au résultat obtenu par chacun des solveurs sur une instance.

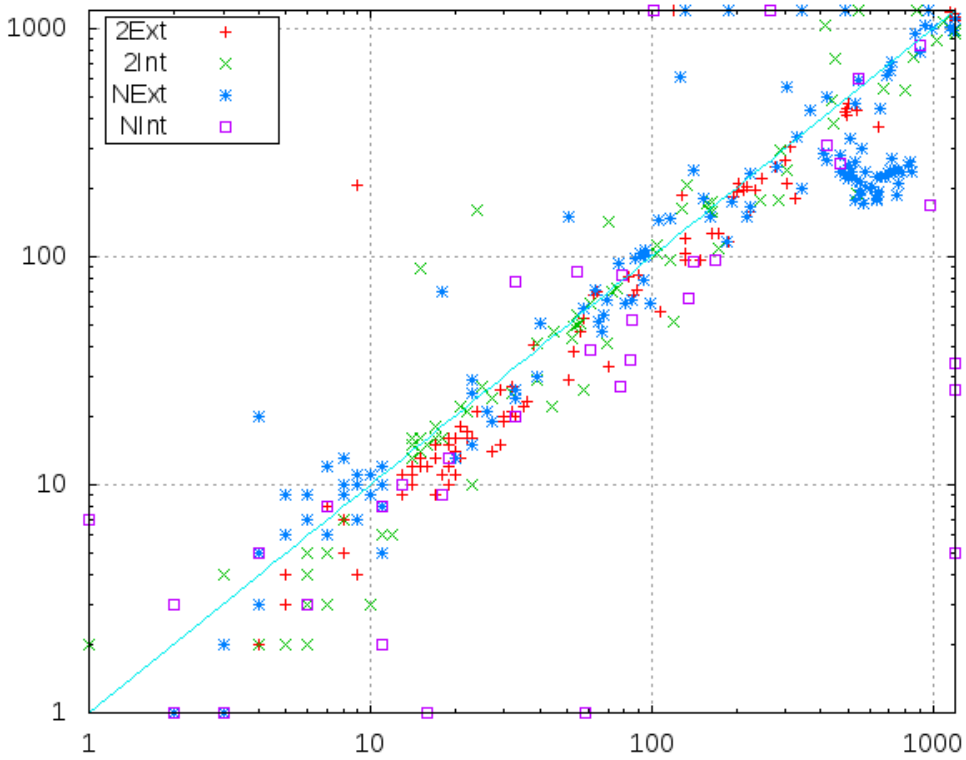


FIGURE 8.14 – Comparaison de FAC-SOLVER et MAC sur les instances insatisfiables. Chaque point (FAC-SOLVER, MAC) reporté correspond au résultat obtenu par chacun des solveurs sur une instance.

En plus de permettre d'estimer si la recherche locale est dans un espace de recherche intéressant, nous pouvons voir sur le tableau 8.5 que l'utilisation des variables FAC permet d'améliorer sensiblement les résultats sur certaines instances. Globalement, l'utilisation des variables FAC permet de résoudre quelques instances en plus (9 instances).

Instance	SAT/UNSAT ?	temps (FAC)	temps (¬ FAC)
uclid-elf-rf8	UNSAT	**305.15**	time out
uclid-37s-smv	UNSAT	**387.50**	659.58
par-16-5	SAT	**168.87**	329.06
primes-10-40-2-7	SAT	**891.01**	time out
primes-20-20-2-7	SAT	976.68	**313.31**
queensKnights-100-5-add	UNSAT	**1,120.18**	time out
queensKnights-100-5-mul	UNSAT	**1,165.81**	time out
queensKnights-80-5-mul	UNSAT	**343.68**	time out
rand-2-40-18	UNSAT	41.47	**1.61**

TABLE 8.5 – Comparaison entre FAC-SOLVER avec choix des premiers points de choix dans l'ensemble des variables FAC ou non.

Une autre manière de valider la robustesse de notre approche est d'étudier l'impact de l'interprétation initiale choisie pour la partie RL. Pour cela, nous avons sélectionné 96 instances (de manière aléatoire) parmi l'ensemble des instances et pour chacune d'entre elles nous avons lancé notre approche 50 fois avec des interprétations initiales différentes. Les résultats montrent que le choix de l'interprétation ne modifie pas outre mesure les résultats obtenus par notre approche. En effet, lorsqu'une instance a été résolue au moins une fois, elle l'a été dans les 49 autres lancements dans 97 % des cas avec un écart moyen de 2.52 secondes.

8.5 Perspectives et Conclusions

Dans ce chapitre, le concept de variables FAC a été introduit et étudié dans le cadre de la résolution du problème CSP. Le but de cette étude a été de développer un solveur hybride tirant parti de manière efficace des différentes approches présentes en son sein. Les résultats obtenus sur un très large panel d'instances ont permis de montrer que le but que nous nous étions fixé à été atteint.

Une question se pose naturellement : dans quelle mesure chacune des composantes de notre approche prend-elle part à l'amélioration de l'efficacité du solveur ? D'après les expérimentations menées, chacune des composantes permettait de trouver la solution et chacune était nécessaires (variables FAC, RL, méthode hybride impliquant la RL et un processus de filtrage) pour assurer la domination de notre méthode. En particulier, nous avons mesuré que dans 56 % des instances considérées, des variables FAC ont été détectées et ont ainsi permis de jouer un rôle prépondérant dans la résolution du problème (même lorsque celui-ci est satisfiable).

L'algorithme FAC-SOLVER proposé est élémentaire et peut être amélioré par un réglage de ses différents paramètres de plusieurs manières. En particulier, une étude expérimentale plus poussée pourrait permettre d'optimiser les différentes variables de contrôle et les différents facteurs d'augmentation ($slsProgress$ et $maxConf$), lesquels ont été fixés arbitrairement. De plus, notre implémentation n'inclut pas certaines des techniques de simplification utilisées habituellement dans le cadre de CSP, comme par exemple l'exploitation des symétries ou des contraintes globales. Nous pensons que l'intégration

de ces techniques au sein de notre solveur permettrait d'augmenter de manière significative ses performances. Il pourrait aussi être intéressant d'explorer une relaxation du concept de variables FAC en prenant en compte (en plus des variables FAC) les variables apparaissant le plus souvent dans les contraintes falsifiées. Cette approche pourrait être utile dans le cas où le CSP considéré ne possède pas de variable FAC (plusieurs noyaux disjoints).

Finalement, nous pensons que le concept de variables FAC est un bon compromis entre : d'une part le faible coup de calcul nécessaire à la RL pour les détecter, et d'autre part, l'apport théorique concernant l'heuristique de choix de variables d'un solveur MAC afin d'obtenir une preuve de petite taille. Prenant part à l'ensemble des MUC du problème, les variables FAC permettent de se focaliser sur la partie difficile du problème. Cependant, il est facile de trouver des instances insatisfiables où les variables FAC ne prennent conceptuellement pas part à la cause réelle de l'insatisfiabilité, mais apparaissent simplement dans l'ensemble des variables de tous les MUC du problème (alors qu'elles ne fournissent pas véritablement d'information sur la cause du conflit). Raffiner le concept de variable FAC afin de capturer plus finement l'essence de l'insatisfiabilité tout en gardant une heuristique efficace (temps de calcul) constitue un véritable challenge.

Conclusion

Dans cette partie, nous avons présenté trois contributions s'inscrivant dans le cadre de la résolution hybride des problèmes SAT et CSP.

La première a donné lieu à l'élaboration du solveur CDLS. Ce solveur, dédié à la résolution de SAT, consiste à ajouter des clauses produites par résolution à la formule afin de s'extraire des minima locaux. Pour cela, nous avons proposé d'adapter le concept d'analyse de conflits à partir d'un graphe d'implications au cadre de la recherche locale. Ce graphe, nommé graphe conflit, est construit à partir d'une interprétation complète en considérant les clauses falsifiées comme les causes de l'échec (minimum local) et les clauses unisatisfaites comme les raisons ayant conduit à cette situation. Pour analyser un tel graphe, nous avons proposé deux approches. La première consiste à considérer les notions de clauses critiques afin de construire un chemin de clauses critiques à partir duquel une nouvelle clause peut être produite par résolution. Les résultats expérimentaux montrent que, bien que ne résolvant pas beaucoup plus d'instances, cette approche permet d'améliorer la vitesse de résolution de la méthode de recherche locale WALKSAT sur laquelle il a été greffé. La seconde approche consiste, quant à elle, à recontruire explicitement le graphe conflit à partir d'une interprétation partielle obtenue par propagation unitaire. Pour cela, nous construisons une nouvelle interprétation partielle à partir de l'interprétation complète courante issue de la recherche locale. Cette interprétation est ensuite utilisée afin de générer et d'analyser un graphe conflit. Nous avons comparé expérimentalement cette approche à différentes méthodes de recherche locale, hybride et CDCL. Sans égaler les approches CDCL sur les instances applicatives, notre approche obtient des résultats très encourageants.

Les travaux effectués autour de CDLS nous ont conduits à proposer SATHYS, un démonstrateur complet hybride combinant recherche locale et solveur CDCL. Ce dernier s'appuie sur un nouveau schéma d'hybridation qui repose sur un échange réciproque d'informations entre les différentes composantes mises en jeu. La particularité de ce schéma est que SATHYS peut être considéré soit comme une méthode de recherche locale utilisant un solveur CDCL afin de s'extirper des minima locaux, soit comme un solveur CDCL employant un solveur de recherche locale afin de se diriger vers les parties difficiles du problème. Nous avons comparé notre technique à différentes approches de recherche locale, hybride et CDCL. Bien que son développement soit récent et loin d'être mature, les résultats obtenus par SATHYS sont particulièrement remarquables. En effet, bien que légèrement moins performant que les solveurs CDCL sur les instances applicatives, notre solveur est très compétitif sur toutes les catégories d'instances. De plus, il a la particularité d'être très robuste puisqu'il résout globalement plus d'instances que les solveurs de l'état de l'art.

La dernière contribution s'intègre, quant à elle, dans le cadre du problème CSP. Les performances remarquables obtenues par le solveur SATHYS nous ont conduits à proposer FAC-SOLVER. Ce dernier étend le schéma d'hybridation du solveur SATHYS et le raffine avec le concept de variable FAC. Afin d'obtenir une méthode hybride efficace, nous avons, dans un premier temps, proposé un solveur de recherche locale, nommé WCSP, intégrant certains des meilleurs critères d'échappement utilisés dans le cadre de SAT. Les expérimentations conduites sur ce dernier montrent qu'il est très performant comparativement à certains solveurs de recherche locale de l'état de l'art. Nous avons ensuite intégré WCSP à notre solveur FAC-SOLVER. La combinaison des deux a permis de générer une méthode qui s'est avérée être très efficace et particulièrement robuste.

Troisième partie

Utilisation de l’heuristique de choix de polarité

Introduction

Comme nous avons pu le voir lors de la description des solveurs CDCL (voir chapitre 3.1.5), les solveurs SAT modernes sont basés sur la combinaison de différentes composantes (heuristique de choix de variable, heuristique de choix de polarité, stratégie de redémarrage, politique de nettoyage de la base de clauses apprises et apprentissage). Parmi ces dernières, l'heuristique de choix de polarité a pour rôle d'aiguiller le solveur dans un espace de recherche. En effet, lorsqu'une nouvelle variable est sélectionnée comme point de choix, l'assigner à vrai ou à faux conduit à choisir l'espace de recherche qui sera examiné en priorité. Partant de ce principe, nous proposons dans cette partie deux contributions utilisant l'heuristique de choix de polarité afin d'obtenir des informations sur l'espace de recherche qui sera potentiellement exploré dans le futur.

La première consiste en une nouvelle heuristique dynamique pour la gestion de la base de clauses apprises dans le cadre des solveurs SAT modernes (Audemard *et al.* 2011a;b). Cette approche est basée sur le principe d'activation et de désactivation de clauses. Étant donnée une étape de la recherche, nous utilisons une fonction pour activer certaines clauses et en désactiver d'autres. Cette fonction, basée sur l'heuristique de choix de polarité *progress saving*, tente d'exploiter certaines informations du passé pour prédire si une clause sera ou non utilisée dans le futur. Notre approche consiste alors à geler une clause quand elle est jugée inutile à la recherche, et à la réactiver lorsqu'elle peut jouer un rôle dans l'établissement de la preuve. Cette stratégie diffère des stratégies habituelles car les clauses ne sont pas directement supprimées.

La seconde contribution s'intègre dans la résolution parallèle du problème SAT à l'aide de solveurs *portfolio* (Guo et Lagniez 2011a;b). Afin d'obtenir une approche portfolio efficace, il est nécessaire que les différents paramètres utilisés pour configurer les solveurs tendent à les rendre complémentaires entre eux. Une des difficultés de ce type d'approche est de paramétrer les différents solveurs de telle manière que deux solveurs n'effectuent pas la même tâche. En effet, dans ce cas, l'une des deux unités de calcul effectue un travail inutile et redondant. Le problème est qu'il est impossible de prévoir, *a priori*, le comportement d'un solveur par rapport à ses paramètres initiaux. Pour pallier ce problème, nous proposons une mesure permettant d'estimer le comportement d'un solveur vis-à-vis d'un autre. Elle consiste à considérer l'heuristique de choix de polarité afin de prédire vers quel espace de recherche se dirige un solveur. De cette manière, deux solveurs peuvent être considérés comme proches s'ils tentent d'explorer le même l'espace de recherche. Cette mesure est ensuite utilisée pendant la recherche pour ajuster dynamiquement l'heuristique de choix de polarité. Cet ajustement a pour but d'éloigner deux solveurs considérés comme trop proches.

Chapitre 9

Vers une gestion fine et dynamique de la base de clauses apprises

Sommaire

DANS CE CHAPITRE, nous proposons une nouvelle stratégie dynamique pour la gestion de la base de clauses apprises dans le cadre des solveurs SAT modernes. Cette approche est basée sur le principe d'activation et de désactivation de clauses. Étant donnée une étape de la recherche, nous utilisons une fonction pour activer certaines clauses et en désactiver d'autres. Cette fonction, basée sur le *progress saving*, tente d'exploiter certaines informations du passé pour prédire si une clause sera ou non utilisée dans le futur. L'idée de notre approche consiste alors à geler une clause quand elle est jugée inutile à la recherche et à la réactiver lorsqu'elle peut jouer un rôle dans l'établissement de la preuve. Cette stratégie diffère des stratégies habituelles dans le fait que les clauses ne sont pas directement supprimées. La plupart des résultats reportés dans ce chapitre ont fait l'objet de publications (Audemard *et al.* 2011a;b).

Ce chapitre est organisé de la manière suivante. Après avoir introduit notre nouvelle mesure, basée sur le *progress saving*, nous présentons une approche dynamique de nettoyage de la base des clauses apprises basée sur la notion d'activation et de désactivation de clauses et étudions expérimentalement son comportement. Avant de conclure et de donner quelques perspectives, nous comparons de manière expérimentale notre politique de nettoyage des clauses apprises avec les politiques de nettoyage de l'état de l'art.

9.1 Une nouvelle mesure pour identifier les bonnes clauses

L'analyse de conflits est une composante importante des solveurs SAT modernes. En effet, d'un point de vue théorique, Pipatsrisawat et Darwiche (2009a) ont récemment prouvé que, même dans le cas où le solveur effectue un redémarrage à chaque conflit, l'apprentissage de clauses tel qu'il est fait dans ce type de solveur permet de simuler la résolution générale. Ce résultat implique que, les solveurs SAT modernes peuvent, d'une certaine manière, être vus comme une méthode de preuve par résolution avec une stratégie

de suppression de clauses. Par conséquent, la complétude des solveurs SAT modernes est fortement liée à la politique de suppression de clauses. Par exemple, supposons que la stratégie de nettoyage de la base de clauses apprises soit très agressive, dans ce cas il est impossible de garantir la complétude du solveur. Par conséquent, définir quelle clause est utile pour amener à la preuve est d'une grande importance pour l'efficacité des solveurs. Cependant, répondre à une telle question est calculatoirement difficile et est très proche du problème qui consiste à trouver une preuve de taille minimale. Pour apporter une solution à ce problème nous définissons une mesure simple destinée à évaluer la pertinence d'une clause et nous montrons expérimentalement son efficacité.

9.1.1 Définition de la mesure PSM

La mesure proposée dans cette section est basée sur la notion de *progress saving* (Pipatsrisawat et Darwiche 2007) habituellement utilisée pour déterminer la valeur de vérité avec laquelle sera affectée la prochaine variable de décision (voir chapitre 3.1.4.3). Cette mesure nommée PSM (*Progress Saving based quality Measure*) est définie de la manière suivante.

Définition 9.1 (mesure PSM)**.** *Étant donnée une clause α et une interprétation complète $\mathcal{P}$ représentant l'ensemble des polarités associé à chaque variable, nous définissons* $\text{PSM}_{\mathcal{P}}(\alpha) = |\mathcal{P} \cap \alpha|$.

Premièrement, il est important de noter que notre mesure PSM est dynamique. En effet, puisque l'ensemble des littéraux $\mathcal{P}$ associé à la polarité des variables sauvegardé évolue durant la recherche, le PSM d'une clause donnée évolue par la même occasion. Par exemple, quand une clause est apprise son PSM est égal à 0, et lorsque le retour arrière est effectué il devient égal à 1. Il est aussi important de noter que lorsqu'une clause est à l'origine de la propagation d'un littéral, son PSM est aussi égal à un. Ces remarques préliminaires suggèrent que les clauses possédant un PSM faible sont plus importantes pour la suite de la recherche. En effet, considérons $\mathcal{I}$ l'interprétation partielle courante, $\mathcal{P}$ l'interprétation complète représentant la sauvegarde courante de l'ensemble des littéraux associés à la polarité avec laquelle a été affectée chaque variable précédemment et α une clause. Comme $\mathcal{I} \subset \mathcal{P}$, $psm_{\mathcal{P}}(\alpha)$ représente le nombre de littéraux affectés à vrai par $\mathcal{I}$ ou qui seront affecté à vrai par $\mathcal{P} \backslash \mathcal{I}$. Par conséquent, une clause avec un petit PSM a de fortes chances d'être utilisée pour la propagation unitaire ou d'être falsifiée. Au contraire, une clause avec un grand PSM a plus de chance d'être satisfaite par plus d'un littéral et par conséquent d'être inutile pour la suite de la recherche. Pour analyser et valider cette hypothèse, un ensemble d'expérimentations ont été conduites.

9.1.2 Études expérimentales de la mesure PSM

La figure 9.1 reporte, pour quelques instances, la moyenne du nombre de fois où une clause avec une certaine valeur de PSM a été utilisée durant le processus de propagation. Dans cette expérimentation, nous considérons comme intervalle de temps le moment de la recherche où la base de clauses apprises est nettoyée. Cette intervalle de temps sera noté t_k avec $k > 0$ (le début de la recherche est en t_0). Soient $\mathcal{P}_{t_k}$ et $\mathcal{P}_{t_{k+1}}$ les interprétations représentant respectivement le *progress saving* aux étapes t_k et t_{k+1}. Considérons la fenêtre de temps entre t_k et t_{k+1}, lorsqu'une clause α, provenant de la base de clauses apprises, est utilisée dans le processus de propagation unitaire, nous calculons $psm = psm_{\mathcal{P}_{t_k}}(\alpha)$ et incrémentons la variable $tab[psm]$ qui correspond au nombre de fois où une clause avec une telle valeur PSM est utilisée pour propager un littéral. Le nombre moyen de fois où une clause possédant un PSM donné (axe-x) est utilisée dans le processus de propagation unitaire (axe-y), correspond alors à $\alpha(psm)$ divisé par le nombre total de fois où la base de clauses apprises a été nettoyée.

Comme il peut être observé sur la Figure 9.1, les clauses apprises possédant une petite valeur de PSM sont plus souvent utilisées dans le processus de propagation que les clauses possédant une grande valeur. Si nous regardons plus en détail, nous pouvons voir que les clauses les plus utilisées ont une valeur de PSM proche de 10. Nous pouvons assurer, puisque l'expérimentation à été conduite sur un très large panel d'instances, que pour la majorité des instances considérées la distribution de la valeur de PSM ressemble aux deux premières courbes de la Figure 9.1.

Cette première expérimentation illustre le fait que les clauses possédant une faible valeur de PSM sont importantes dans le processus de propagation. Pour valider de manière expérimentale notre mesure nous avons choisi de l'intégrer dans une politique de réduction de la base de clauses apprises (fonction `reductionClausesApprises(`Δ`)` - ligne 15 de l'algorithme 3.4) et de comparer celle-ci avec les approches de réduction de clauses apprises existantes dans la littérature. Cette nouvelle stratégie de réduction a été introduite dans le solveur MINISAT 2.2. Comme pour les approches classiques de réduction, l'ensemble des clauses apprises est tout d'abord trié par ordre croissant (ici, en fonction de leur PSM). Lorsque deux clauses possèdent le même PSM, nous utilisons la stratégie classique basée sur l'activité (VSIDS) pour les départager. Une fois les clauses triées, la base de clauses apprises est réduite de moitié. Comme pour les autres stratégies, nous conservons toujours les clauses binaires.

Dans la suite, toutes nos expérimentations ont été conduites sur un Quad-core Intel XEON X5550 avec 32Gb de mémoire. 292 instances ont été utilisées, elles sont issues de la compétition SAT 2009, catégorie industrielle (Le Berre et Roussel 2009). Pour l'ensemble des expérimentations le temps CPU est limité à 900 secondes.

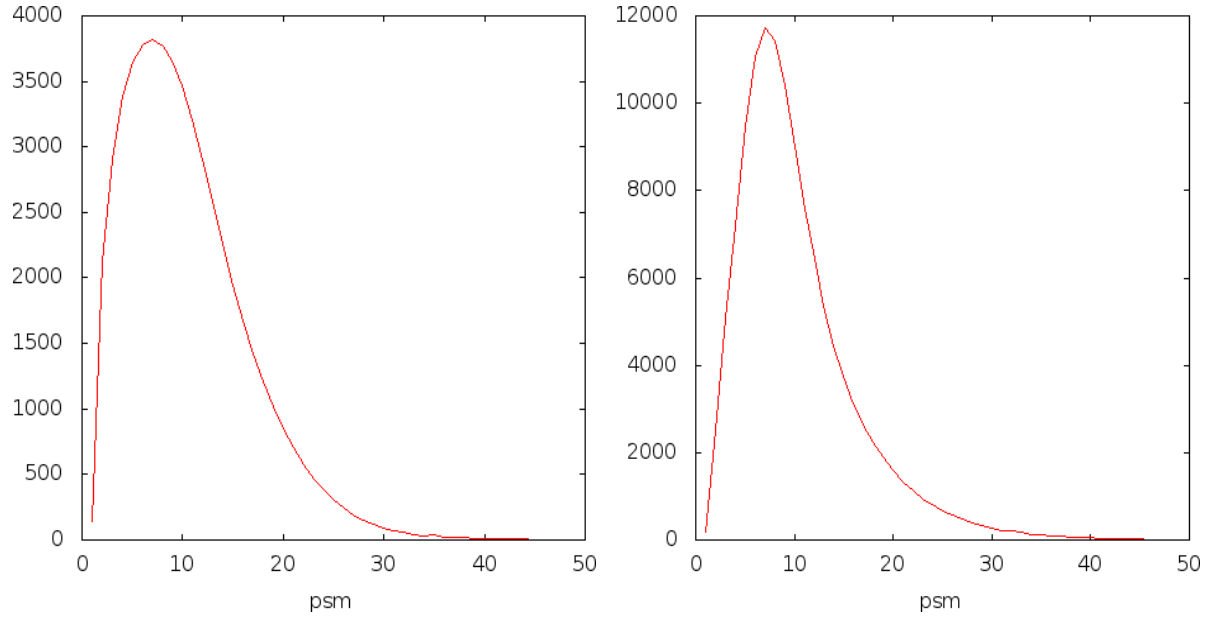
Pour chaque solveur, nous indiquons le nombre d'instances résolues (TOT) ainsi que le nombre d'instances résolues satisfiables (SAT) et insatisfiables (UNS). Nous donnons aussi la moyenne de temps nécessaire pour résoudre une instance (tps moy).

Nous évaluons dans un premier temps les résultats obtenus avec la séquence de réductions utilisée par défaut dans le solveur MINISAT (voir sous-section 3.1.5.2). Le tableau 9.1 résume les résultats obtenus par MINISAT^d (Sörensson et Eén 2009), MINISAT^d+LBD (Audemard et Simon 2009a), MINISAT^d+$taille$ qui utilise la taille des clauses pour trier la base et MINISAT^d + PSM. Comme nous pouvons le constater, MINISAT^d+PSM est le solveur qui résout le plus d'instances. C'est également le meilleur sur les instances SAT. De plus, les faibles performances de MINISAT^d + $taille$ démontrent que les clauses conservées par notre approche ne sont pas uniquement celles de petites tailles. Cette expérimentation montre l'efficacité de notre mesure dans le cas d'une politique utilisant la séquence de réduction proposée par défaut dans MINISAT 2.2.

	Application			
Solveur	SAT	UNS	TOT	tps moy
MINISAT^d	68	**106**	174	142
MINISAT^d + PSM	**73**	104	**177**	130
MINISAT^d + LBD	71	102	173	132
MINISAT^d + $taille$	67	97	164	153

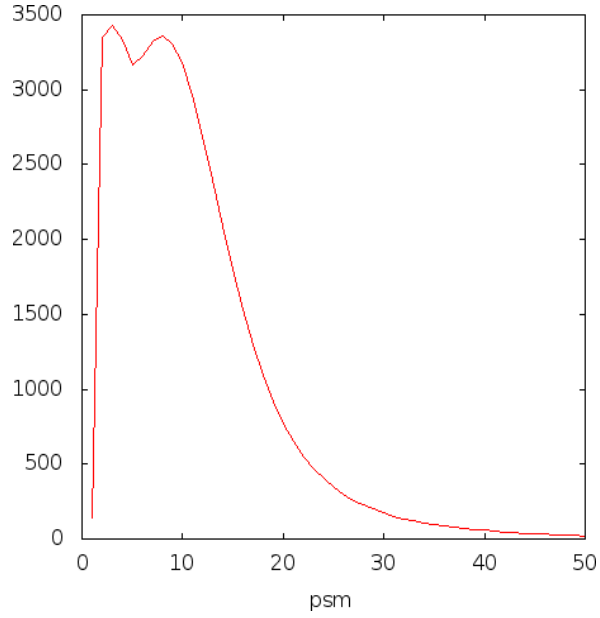
TABLE 9.1 – Résultats obtenus avec la séquence de réduction utilisée par défaut dans le solveur MINISAT. Nous reportons le nombre d'instances satisfaisables, insatisfaisables et totales résolues.

Pour être plus juste dans la comparaison des quatre approches, nous présentons aussi dans le tableau 9.2 les résultats obtenus lorsque la séquence de réduction est plus agressive, comme celle utilisée dans Audemard et Simon (2009a) (noté MINISAT^a). Dans cette expérimentation la séquence de réduction de

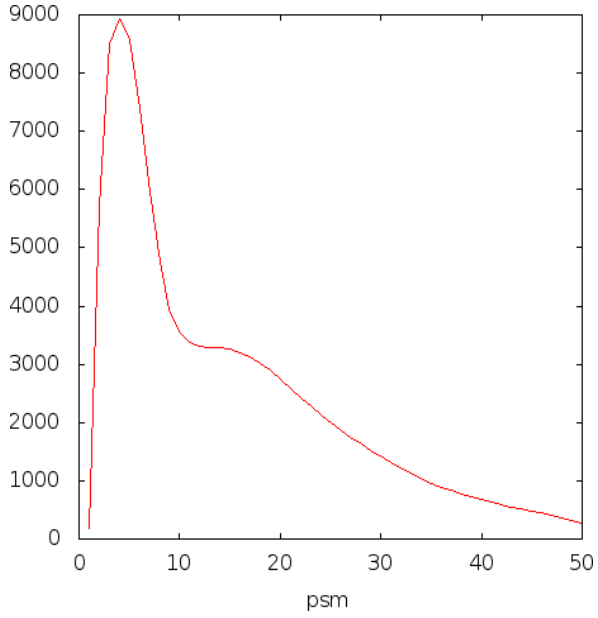


(a) Aprove07-25

(b) q_query_3_l48_lambda



(c) goldb-heqc-frg1mul



(d) countbitsarray08_32

FIGURE 9.1 – Corrélation entre la valeur de PSM d'une clause et son utilisation dans le processus de propagation unitaire.

la base de clauses apprises Δ est calculée de la manière suivante : $t_0 = 4000$ conflits et $t_k = t_{k-1} + 300$ conflits pour $k > 0$. En utilisant une stratégie de réduction plus agressive, les résultats obtenus par le LBD sont meilleurs que ceux obtenus à l'aide de la stratégie VSIDS et par notre mesure PSM. En ce qui concerne la stratégie basée sur la taille des clauses, nous pouvons voir que ce seul critère ne permet pas d'identifier les bonnes clauses.

	Application			
Solveur	SAT	UNS	TOT	tps moy
MINISAT^a	68	94	162	136
MINISAT^a + PSM	70	93	163	140
MINISAT^a + LBD	**72**	**96**	**168**	128
MINISAT^a + $taille$	61	76	137	180

TABLE 9.2 – Résultats avec la séquence de réduction utilisée agressive.

Pour résumer ces premières expérimentations, nous reportons dans le tableau 9.3 les résultats obtenus par les différentes versions de MINISAT testées précédemment sur une sélection d'instances (PSM^d correspond à MINISAT^a + PSM, LBD^d correspond à MINISAT^a + LBD, *etc.*).

	SAT	MINISAT^d	PSM^d	LBD^d	MINISAT^a	PSM^a	LBD^a
emptyroom-4-h21-unsat.cnf	N	236	**204**	232	241	**168**	262
eq.atree.braun.10.unsat.cnf	N	–	**751**	–	345	396	**289**
partial-10-15-s.cnf	Y	–	–	–	686	**419**	–
partial-5-17-s.cnf	Y	–	–	–	642	707	**417**
partial-10-13-s.cnf	Y	–	–	–	–	**448**	–
UR-15-10p1.cnf	Y	810	**757**	–	–	–	–
UTI-15-10p1.cnf	Y	96	**94**	100	261	233	**163**
UR-20-5p1.cnf	Y	489	**405**	544	**774**	810	–
UCG-15-10p0.cnf	N	274	**272**	325	719	400	**385**
UTI-15-5p0.cnf	N	371	**339**	374	**292**	302	327
UR-15-10p0.cnf	N	919	**779**	–	–	–	**513**
UR-15-5p0.cnf	N	57	**54**	58	**66**	67	92
UTI-15-5p1.cnf	Y	50	**45**	53	90	**56**	95
UTI-20-10p1.cnf	Y	–	**824**	–	–	–	–
UR-20-5p0.cnf	N	873	834	**804**	–	–	–
ACG-15-10p0.cnf	N	**709**	**709**	907	–	**651**	–
q_query_3_l45_lambda.cnf	N	126	**89**	117	**146**	300	255
q_query_3_L90_coli.sat.cnf	N	71	**66**	72	200	**101**	111
q_query_3_L100_coli.sat.cnf	N	51	**48**	57	226	**91**	311
q_query_3_l44_lambda.cnf	N	**107**	109	110	**130**	148	215

TABLE 9.3 – Résultats obtenus par notre approche sur une sélection d'instances insatisfiables. Les temps reportés dans ce tableau sont exprimés en secondes.

9.2 Geler pour ne pas oublier : une politique dynamique de réduction de la base de clauses apprises

Dans la section précédente, nous avons défini une nouvelle mesure basée sur la notion de *progress saving* pour identifier les clauses apprises importantes. Dans cette section, nous décrivons une approche dynamique de gestion de la base de clauses apprises. Cette approche est basée sur deux notions-clés. Premièrement, le *progress saving* évolue durant la recherche. Par conséquent, une clause peut être considérée comme inutile (avec une grande valeur de PSM) à un instant de la recherche et devenir intéressante (avec une petite valeur de PSM) à un autre moment de la recherche. Deuxièmement, déterminer si une clause donnée est importante pour la preuve est un problème calculatoirement difficile. Toutes les poli-

tiques de nettoyage de la base de clauses apprises proposées dans la littérature ne sont pas certaines de ne pas supprimer une clause importante pour la suite de la recherche. Pour ces deux raisons, l'approche proposée introduit un nouveau concept qui consiste à geler certaines clauses. Lorsqu'une clause est considérée comme inutile à un instant de la recherche celle-ci devient gelée et est réactivée lorsqu'elle est de nouveau considérée comme importante. Plus précisément, geler (respectivement activer) une clause signifie que la clause est déconnectée (respectivement connectée) à la base des clauses apprises et ne participe donc pas à la recherche (propagation unitaire ...).

Cette politique de nettoyage de la base de clauses apprises basée sur le principe de désactivation et d'activation de clauses ne peut pas être utilisé avec les mesures habituelles. En effet, le LBD d'une clause apprise est statique car il a été défini à la création de la clause, et ne change pas durant la recherche. Tandis que la mesure basée sur l'activité (basée sur VSIDS) est dynamique mais peut seulement mettre à jour les clauses attachées (les clauses doivent être utilisées pour être pondérées).

9.2.1 Politique de gestion de la base de clauses apprises

À présent, décrivons de manière formelle notre politique de nettoyage de la base de clauses apprises. Premièrement, comme le PSM d'une clause est fortement dynamique, nous introduisons la notion de *déviation* entre deux *progress saving* successifs.

Définition 9.2 (déviation)**.** *Soit V_{t_k} l'ensemble des variables assignées par le solveur entre l'étape t_{k-1} et t_k. La déviation d_{t_k} est définie de la manière suivante :*

$$d_{t_k} = \frac{\delta_{\mathfrak{h}}(\mathcal{P}_{t_k}, \mathcal{P}_{t_{k-1}})}{|V_{t_k}|}.$$

Cette notion de déviation est une normalisation de la distance de Hamming pour ne prendre en compte que les variables réellement utilisées par le solveur. Elle permet de calculer l'évolution de l'interprétation liée au *progress saving* entre deux nettoyages de base successifs. Une déviation qui tend vers zéro indique que le solveur continue d'explorer le même espace de recherche, alors qu'une déviation importante indique que le solveur explore une partie différente de l'espace de recherche.

Pour obtenir une vue plus précise du comportement de la déviation, nous introduisons la notion de déviation minimale d^m.

Définition 9.3 (déviation minimale)**.** *À l'instant t_k la déviation minimale est donnée par :*

$$d^m_{t_k} = min\{d_{t_i} | 0 \leq i \leq k\}.$$

L'utilisation de cette notion de déviation minimale nous permet d'affiner notre mesure PSM. En effet, soit α une clause évaluée à l'étape t_k, si $\text{PSM}_{\mathcal{P}_{t_k}}(\alpha) > d^m_{t_k} \times |\alpha|$ alors la clause α a de fortes chances d'être satisfaite dans le futur, sinon elle sera sûrement utilisée dans le processus de propagation.

La Figure 9.2 décrit notre approche sous la forme d'un automate d'états. À chaque étape de nettoyage t_k, les clauses peuvent passer d'un état à un autre suivant certaines conditions. Pour commencer, notons qu'une clause apprise α possède trois états :

1. *état activé* $\mathcal{A}$: α est activée et attachée, elle est utilisée par le solveur ;
2. *état gelé* $\mathcal{F}$: α est gelée, c'est-à-dire que α n'est pas attachée ;
3. *état mort* $\mathcal{D}$: α est supprimée.

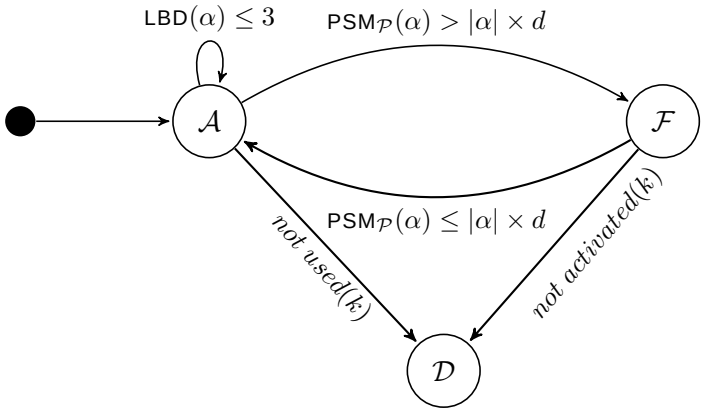


FIGURE 9.2 – Cycle de vie d'une clause apprise.

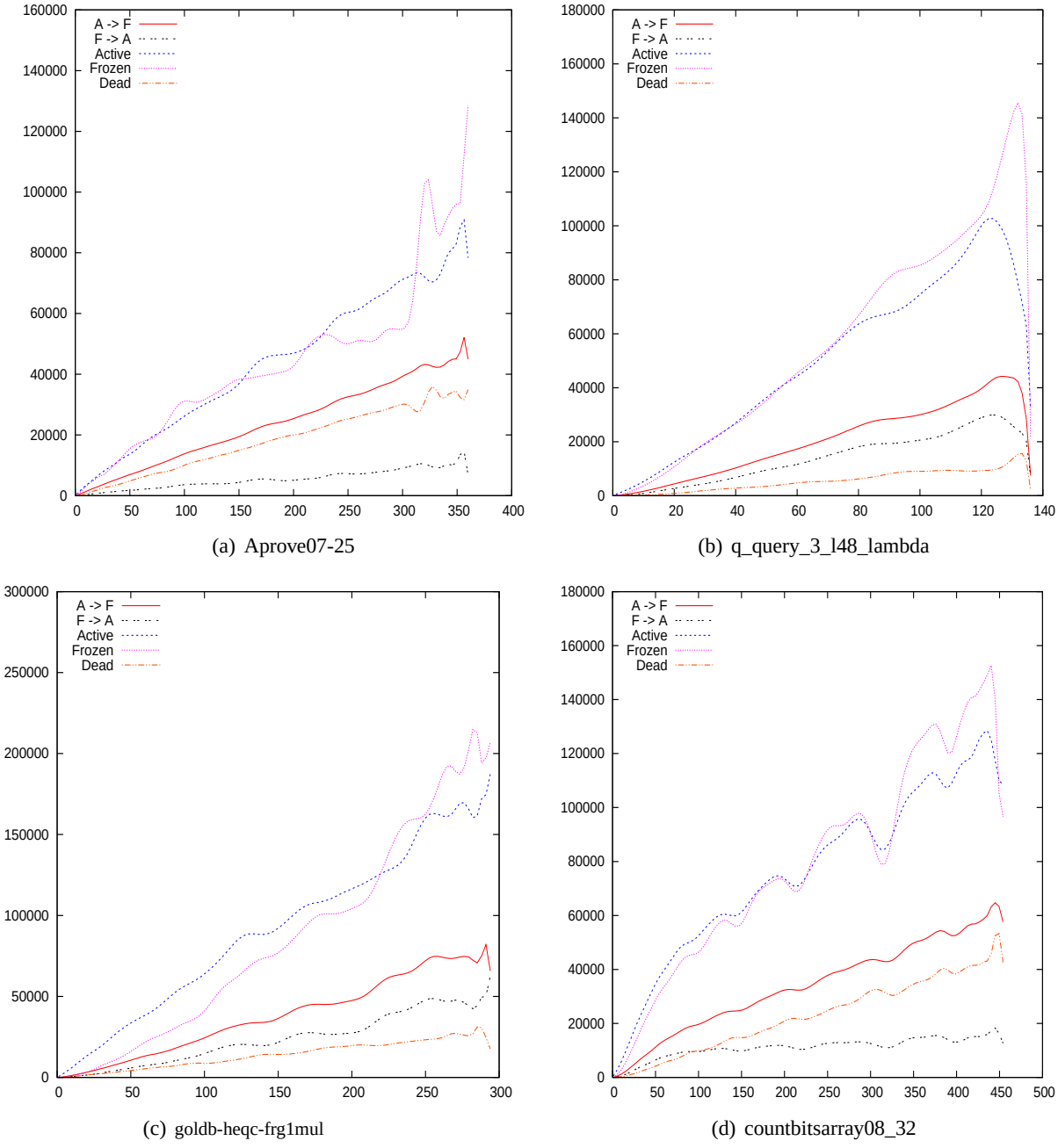
Nous décrivons à présent les différentes transitions de l'automate à états :

- chaque fois qu'une nouvelle clause est apprise celle-ci entre dans l'état $\mathcal{A}$;
- une clause $\alpha \in \mathcal{A}$ avec une valeur de LBD ($lbd(\alpha) \leq 3$ sur la figure) reste dans l'état $\mathcal{A}$ jusqu'à la fin du processus de recherche ;
- une clause $\alpha \in \mathcal{A}$ telle que $\frac{\text{PSM}_{\mathcal{P}_{t_k}}(\alpha)}{|\alpha|} > d^m_{t_k}$ passe à l'état gelé $\mathcal{F}$;
- une clause $\alpha \in \mathcal{F}$ telle que $\frac{\text{PSM}_{S_{n_i}}(\alpha)}{|\alpha|} \leq d^m_{n_i}$ passe à l'état activé $\mathcal{A}$;
- une clause $\alpha \in \mathcal{F}$ qui n'a pas été activée pendant k étapes de nettoyage est supprimée. De manière similaire, une clause $\alpha \in \mathcal{A}$ restant active pendant plus de k étapes sans participer à la recherche est aussi supprimée. Dans les deux cas, cette clause entre dans l'état $\mathcal{D}$. Pour notre étude expérimentale la valeur de k a été fixée empiriquement à 7.

Un des principaux avantages de notre méthode réside dans le fait qu'il est possible d'augmenter la fréquence de nettoyage de la base de clauses apprises sans l'inconvénient de supprimer une clause importante pour la suite de la recherche. Nous pouvons donc utiliser une politique de nettoyage de base très agressive. Nous avons choisi pour les expérimentations présentées dans la partie suivante $t_0 = 500$ et $t_{k+1} + 100$ conflits.

9.2.2 Étude du cycle de vie d'une clause apprise vis-à-vis de notre schéma

Pour valider notre approche, nous avons étudié expérimentalement le comportement des clauses au sein de l'automate à état. La Figure 9.3 montre, pour les mêmes instances que celles considérées dans la Figure 9.1, le nombre de clauses actives, gelées et supprimées ainsi que le nombre de transitions de l'état actif à l'état gelé et vice versa. Ces différentes données sont représentées sur l'axe y des ordonnées tandis que sur l'axe des abscisses x représente le nombre d'opérations de nettoyage de la base de clauses apprises. Pour des raisons de lisibilité, l'ensemble des courbes a été lissé. Pour l'ensemble de ces instances, le nombre de clauses gelées ($Frozen$) et le nombre de clauses actives ($Active$) sont relativement similaires. En ce qui concerne le taux de transfert d'un état à un autre, nous pouvons remarquer que le nombre de clauses passant de l'état actif à l'état gelé ($\mathcal{A} \rightarrow \mathcal{F}$) est plus grand que le nombre de clauses passant de l'état gelé à l'état actif ($\mathcal{F} \rightarrow \mathcal{A}$). Ceci s'explique par le fait que la base des clauses apprises est sans cesse alimentée par de nouvelles clauses (ces clauses sont initialement dans l'état actif $\mathcal{A}$). Pour terminer, nous pouvons observer qu'à chaque étape du nettoyage certaines clauses sont définitivement supprimées ($Dead$).



(a) Aprove07-25

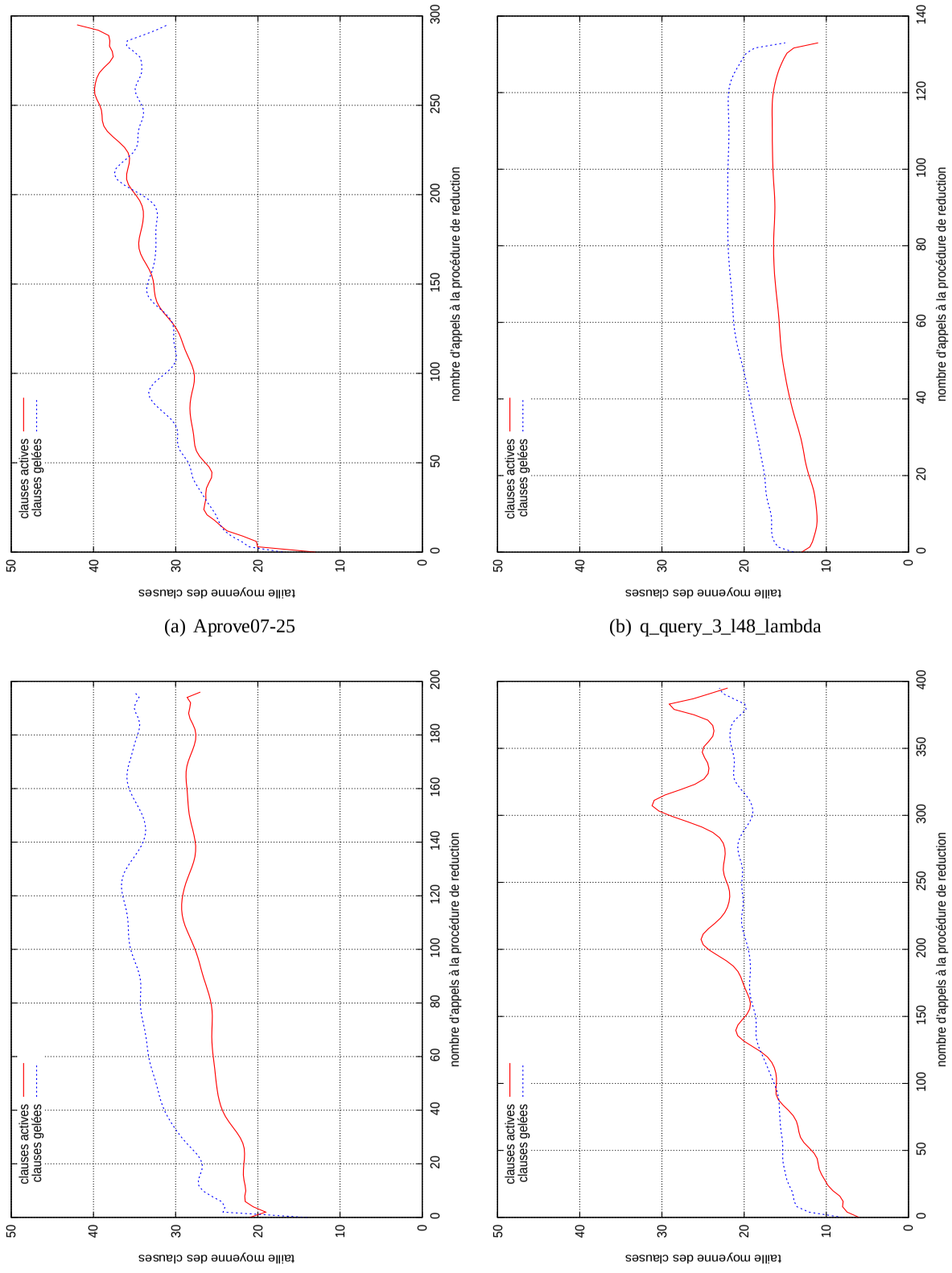
(b) q_query_3_l48_lambda

(c) goldb-heqc-frg1mul

(d) countbitsarray08_32

FIGURE 9.3 – Étude du taux de transfert.

Afin de vérifier que les clauses conservées par notre schéma ne sont pas uniquement celles de petite taille, nous avons étudié la taille moyenne des clauses actives et gelées après chaque appel de la fonction de nettoyage de la base de clause apprises. Les résultats obtenus sur les quatre instances considérées précédemment sont reportés sur la figure 9.4. L'axe des ordonnées donne la taille moyenne des clauses gelées (trait en pointillé) et actives (trait plein) au i^{ème} appel de la procédure de réduction (axe des abscisses). Cette figure montre clairement que notre approche ne privilégie pas systématiquement les clauses de petites taille et que la distribution des clauses entre les deux états est assez homogène.



(a) Aprove07-25

(b) q_query_3_l48_lambda

(c) goldb-heqc-frg1mul

(d) countbitsarray08_32

FIGURE 9.4 – Taille moyenne des clauses actives et gelées.

Pour terminer cette étude, nous avons tenté de quantifier l'erreur commise par notre approche, c'est-à-dire le nombre de clauses gelées permettant de propager un littéral. Pour cela, nous avons exécuté notre approche sur l'ensemble des instances et nous avons après chaque appel de la fonction propagation comptabilisé le nombre de clauses dont l'état gelé permet soit de propager un littéral soit de produire un conflit. Les résultats obtenus lors de cette étude sont encourageants, puisque le pourcentage moyen d'erreur est inférieur à 1%.

9.3 Expérimentations

Cette dernière section se divise en deux parties. Dans la première, nous comparons notre stratégie de réduction avec celles de l'état de l'art. Dans la seconde, nous comparons le solveur MINISAT où est implanté notre schéma avec les solveurs de l'état de l'art.

9.3.1 Comparaison avec différentes stratégie de réduction

Nous comparons notre politique dynamique de nettoyage de la base de clauses apprises, appelée MINISAT-psm_{dyn}, avec MINISAT classique, et MINISAT avec la stratégie de réduction basée sur le PSM (MINISAT-PSM) et sur le LBD (MINISAT-LBD) (comme pour la section 3). La figure 9.5 contient les courbes permettant de comparer le nombre d'instances résolues en fonction du temps pour les quatre solveurs.

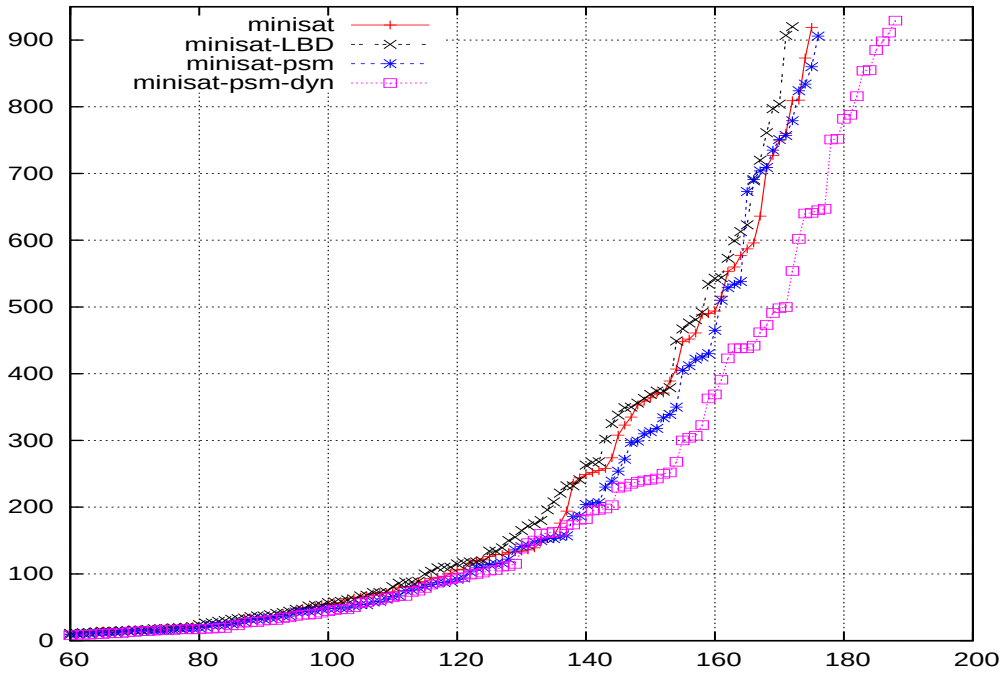
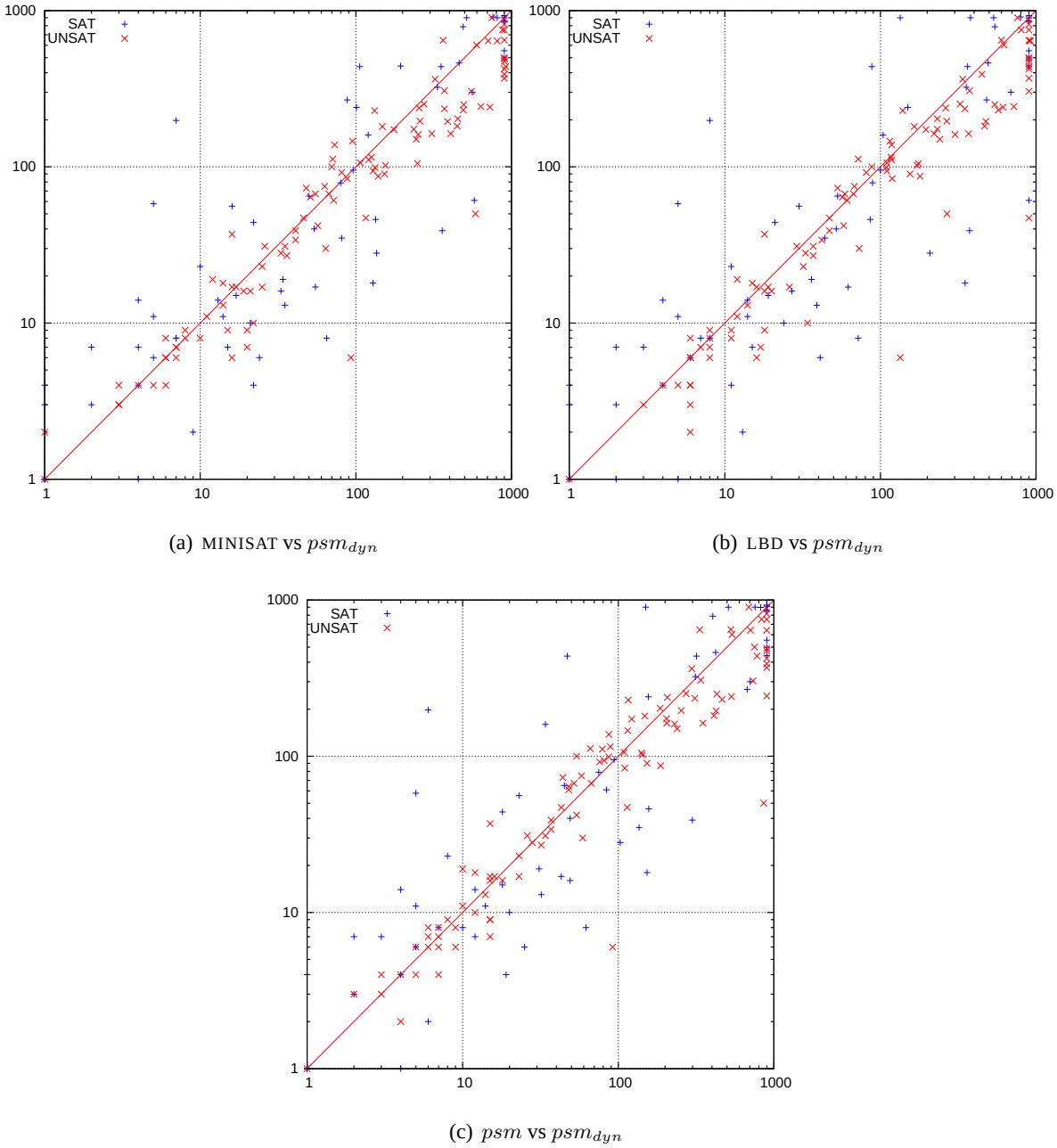


FIGURE 9.5 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour différentes stratégies de nettoyage.

Dans la Figure 9.6 nous reportons les nuages de points correspondant aux comparaisons de MINISAT-psm_{dyn} avec les trois autres solveurs. Chaque point correspond à une instance. Un point en dessous de la

diagonale signifie que cette instance a été résolue plus rapidement avec MINISAT-psm_{dyn}. Les instances SAT et UNSAT sont respectivement représentées sur la figure par le signe plus ($+$) et multiplié ($\times$).



(a) MINISAT vs psm_{dyn}

(b) LBD vs psm_{dyn}

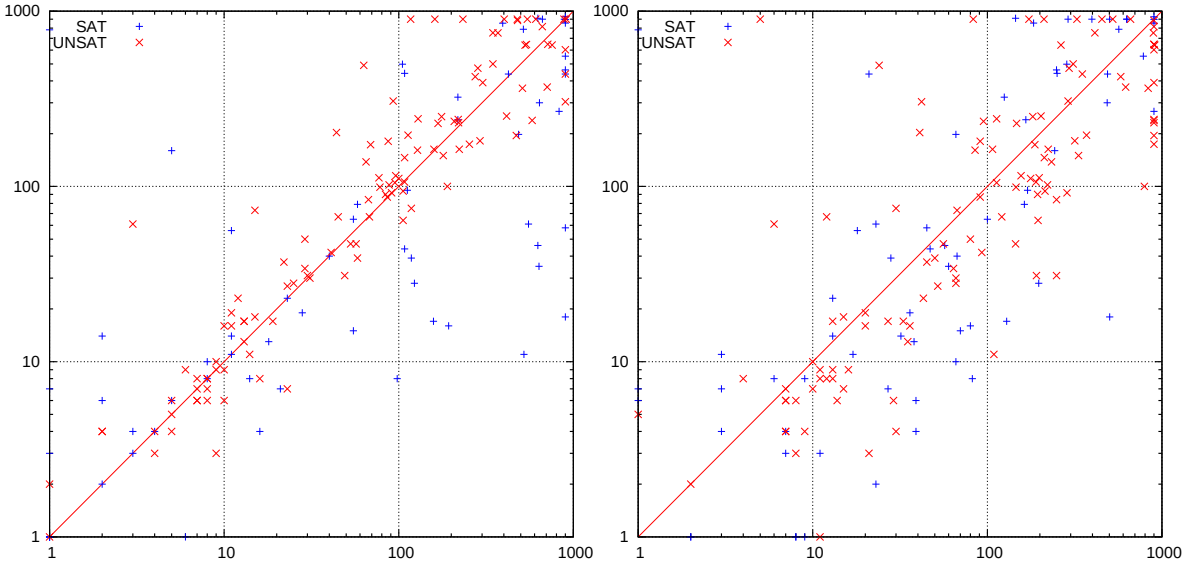
(c) psm vs psm_{dyn}

FIGURE 9.6 – Comparaison avec différentes stratégies de nettoyage de la base de clauses apprises.

Il est clair que notre approche de réduction de la base des clauses apprises permettant de geler des clauses est nettement meilleure que les trois autres. Le solveur résout 189 instances (76 SAT et 113 UNSAT), ce qui est meilleur que les autres solveurs (voir tableau 9.1). De plus, comme nous pouvons le voir sur le nuage de point, MINISAT-psm_{dyn} résout les instances plus rapidement que les autres solveurs.

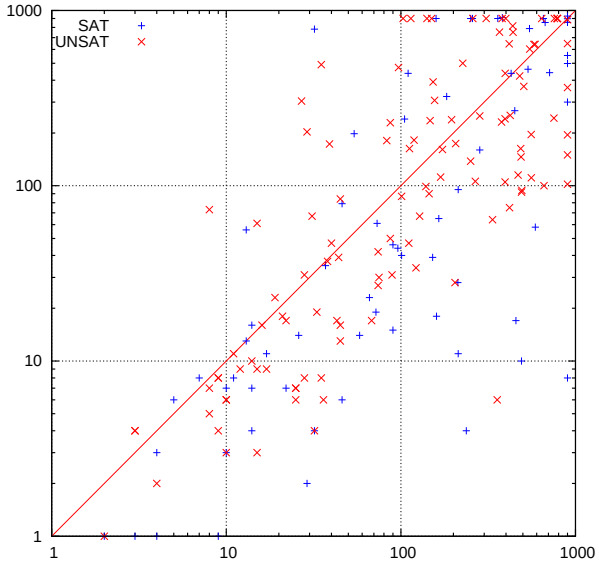
9.3.2 Comparaison avec les solveurs de l'état de l'art

La figure 9.7 reporte les résultats obtenus par notre approche basée sur la notion de réduction dynamique de la base de clauses apprises vis-à-vis des meilleurs solveurs de l'état de l'art. L'analyse des différents nuages de points montre que notre approche, bien qu'elle n'utilise aucune des dernières améliorations proposées dernièrement (exemple : redémarrage dynamique, clause bloquée, *etc.*), est très compétitive.



(a) GLUCOSE vs psm_{dyn}

(b) LINGELING vs psm_{dyn}



(c) CRYPTOMINISAT vs psm_{dyn}

FIGURE 9.7 – Comparaison avec les solveurs de l'état de l'art : GLUCOSE, LINGELING et CRYPTOMINISAT.

Nous reportons dans la figure 9.8 les courbes permettant de comparer le nombre d'instances résolues en fonction du temps pour les quatre solveurs. Les résultats obtenus sont : LINGELING résout 187 instances (77 SAT et 110 UNSAT), GLUCOSE 189 instances (70 SAT et 119 UNSAT) et CRYPTOMINISAT 194 instances (74 SAT et 120 UNSAT). Ces courbes montrent que notre approche résout pratiquement autant d'instances (rappelons que notre solveur résout 189 instances (76 SAT et 113 UNSAT)) que LINGELING et GLUCOSE et légèrement moins d'instances que CRYPTOMINISAT. Nous pouvons remarquer, sur les tableaux 9.4 et 9.5, que notre solveur n'est pas performant que sur certaines classes d'instances. En effet, nous pouvons voir dans le tableau 9.4 que notre solveur ne résout qu'une seule instance de la catégorie *vliw* tandis que les autres approches en résolvent au moins cinq.

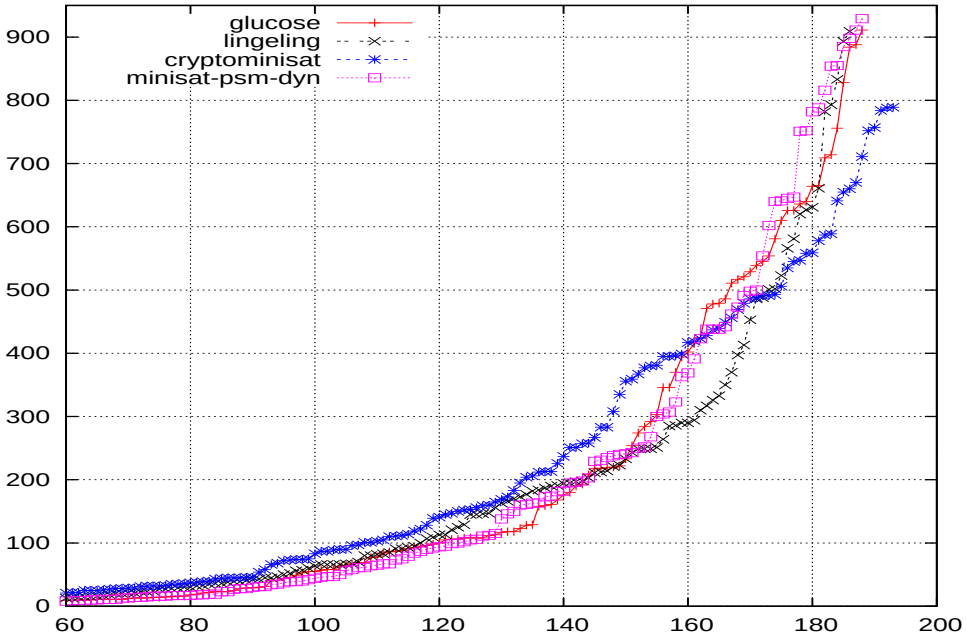


FIGURE 9.8 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) par notre approche et les solveurs de l'état de l'art : GLUCOSE, LINGELING et CRYPTOMINISAT.

	SAT	psm_{dyn}	GLUCOSE	LINGELING	CRYPTOMINISAT
9dlx_vliw_at_b_iq6	N	–	–	–	**757**
9dlx_vliw_at_b_iq3	N	–	233	173	**141**
9dlx_vliw_at_b_iq1	N	491	63	**24**	35
9dlx_vliw_at_b_iq4	N	–	478	326	**258**
9dlx_vliw_at_b_iq2	N	–	117	**83**	103
9dlx_vliw_at_b_iq5	N	–	610	523	**399**
velev-vliw-uns-4.0-9C1	N	–	**161**	661	380
velev-pipe-uns-1.0-8	N	–	–	453	**150**
goldb-heqc-frg1mul	N	–	402	–	**114**
goldb-heqc-x1mul	N	–	911	–	**641**

TABLE 9.4 – Résultats obtenus par psm_{dyn}, GLUCOSE, LINGELING et CRYPTOMINISAT sur une sélection d'instances insatisfiables. Les temps reportés dans ce tableau sont exprimés en secondes.

	SAT	psm_{dyn}	GLUCOSE	LINGELING	CRYPTOMINISAT
q_query_3_l48_lambda	N	146	**108**	212	488
q_query_3_l47_lambda	N	**94**	106	214	493
q_query_3_l46_lambda	N	111	**100**	177	559
q_query_3_L70_coli.sat	Y	**40**	**40**	67	101
q_query_3_L80_coli.sat	N	**47**	57	145	111
q_query_3_L200_coli.sat	N	196	**113**	370	558
q_query_3_L150_coli.sat	N	92	**91**	286	491
q_query_3_l45_lambda	N	115	**96**	156	469
q_query_3_L90_coli.sat	N	112	**77**	198	169
q_query_3_L60_coli.sat	Y	14	**11**	32	58
q_query_3_L100_coli.sat	N	**64**	106	195	335
q_query_3_l44_lambda	N	**106**	108	190	267
ndhf_xits_22_SAT	Y	4	4	**3**	237
ndhf_xits_20_SAT	Y	58	–	**45**	589
rbcl_xits_07_UNSAT	N	**163**	222	223	486
ndhf_xits_21_SAT	Y	11	521	**3**	213
rpoc_xits_07_UNSAT	N	105	**95**	113	395
rpoc_xits_17_SAT	Y	**1**	**1**	2	4
rbcl_xits_06_UNSAT	N	**7**	23	10	25
vmpc_29	Y	**268**	828	–	449
vmpc_30	Y	61	554	**23**	73
vmpc_28	Y	160	**5**	243	283
9vliw_m_9stages_iq3_C1_bug8	Y	**17**	158	129	456
9vliw_m_9stages_iq3_C1_bug5	Y	10	**8**	66	491
gus-md5-10	N	602	–	–	**547**
gus-md5-09	N	238	581	–	**195**
gus-md5-06	N	**11**	14	109	**11**
gss-17-s100	Y	**28**	123	197	212
gss-16-s100	Y	**44**	108	47	96
gss-15-s100	Y	**7**	21	27	14
gss-14-s100	Y	14	**2**	13	26
gss-19-s100	Y	442	**108**	251	711
gss-20-s100	Y	**554**	–	782	–
maxxororand032	N	**647**	714	–	–
velev-pipe-sat-1.0-b10	Y	**8**	98	82	–
manol-pipe-c10nid_i	Y	102	**88**	221	–
manol-pipe-c10nidw	N	**150**	180	333	–
schup-l2s-bc56s-1-k391	N	**363**	511	833	–
ACG-15-10p1	Y	855	–	**184**	–
UTI-20-10p1	Y	**898**	–	–	–
AProVE09-06	Y	**300**	640	486	–
post-cbmc-aes-ee-r3-noholes	N	885	**479**	–	–
cube-11-h13-unsat	N	**195**	471	–	–

TABLE 9.5 – Résultats obtenus par psm_{dyn}, GLUCOSE, LINGELING et CRYPTOMINISAT sur une sélection d'instances. Les temps reportés dans ce tableau sont exprimés en secondes.

9.4 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle mesure permettant d'identifier les clauses importantes pour la suite de la recherche. Cette mesure est dynamique (contrairement à la mesure basée sur le LBD) et peut être calculée sur des clauses ne participant pas à la recherche (contrairement à la mesure basée sur VSIDS). Grâce à ces propriétés, une nouvelle stratégie dynamique de réduction de la base de clauses apprises a été proposée. Cette dernière est basée sur l'activation et la désactivation de clauses, alors qu'avec les stratégies de réduction de l'état de l'art les clauses sont définitivement supprimées. L'ensemble des expérimentations que nous avons conduit ont permis de mettre en exergue un flux bidirectionnel de clauses entre les deux états principaux de notre schéma (c'est-à-dire les états actifs et gelés). Nous avons aussi démontré expérimentalement que les clauses gelées ne servent pas à la propagation unitaire pendant le laps de temps où elles sont inactives (moins de 1% de taux d'erreur). L'implantation de notre schéma au sein du solveur MINISAT a permis d'améliorer sensiblement les performances de ce dernier. De plus, nous avons vu que ce solveur permettait d'obtenir des résultats quasiment similaires aux solveurs de l'état de l'art.

En perspective, nous comptons étudier plus finement l'évolution de l'interprétation $\mathcal{P}$, ce qui nous permettrait de décider quand une réduction de la base de connaissance est souhaitable. De plus, comme la notion de *progress saving* a déjà été utilisée dans d'autres travaux (Biere 2008a, Pipatsrisawat et Darwiche 2009b) nous pensons qu'il serait intéressant de combiner ces différentes approches.

Chapitre 10

Ajustement dynamique de l'heuristique de polarité dans le cadre d'un solveur SAT parallèle

Sommaire

DANS CE CHAPITRE, nous proposons une nouvelle heuristique pour la polarité d'affectation d'une variable, dans le cadre d'un solveur SAT parallèle (voir chapitre 5). La polarité selon laquelle le prochain point de choix est affecté est un processus important des solveurs SAT modernes, en particulier pour les solveurs de type *portfolio*. En effet, ces solveurs sont souvent basés sur une approche de type coopération/compétition. Par conséquent, afin d'obtenir une approche *portfolio* efficace, il est nécessaire que les différents paramètres utilisés pour configurer les solveurs tendent à rendre les solveurs complémentaires entre eux. Néanmoins, paramétrer les différents solveurs de telle manière que deux solveurs n'effectuent pas la même tâche est très difficile à réaliser en pratique. En effet, il est impossible de prévoir, *a priori*, le comportement d'un solveur par rapport à ses paramètres initiaux. Par conséquent, il est difficile de prédire si une des deux unités de calcul ne va pas effectuer un travail inutile et redondant. Pour pallier ce problème, nous proposons une mesure permettant d'estimer le comportement d'un solveur vis-à-vis d'un autre. Elle consiste à considérer l'heuristique de choix de polarité afin de prédire vers quel espace de recherche se dirige un solveur. De cette manière, deux solveurs peuvent être considérés comme proches s'ils tentent d'explorer le même espace de recherche.

Une fois notre mesure définie, nous étudions de manière expérimentale la distance entre les différents solveurs présents dans MANYSAT 1.1. Après analyse des résultats nous proposons un schéma d'ajustement de l'heuristique de polarité. Finalement, nous étudions expérimentalement l'impact de l'application d'un tel schéma dans le cas de trois approches *portfolio* (MANYSAT 1.1, solveur *portfolio* avec des solveurs possédant une architecture identique et MANYSAT 1.5).

Cette contribution a donné lieu à deux publications Guo et Lagniez (2011a;b).

10.1 Estimer la distance entre deux solveurs

Comme précisé précédemment, les solveurs parallèles de type *portfolio* sont souvent basés sur le principe de compétition/coopération. Ainsi dans le solveur MANYSAT, la phase de coopération peut être

assimilée au transfert de clauses apprises. Tandis que la phase de compétition peut être associée aux différentes stratégies choisies (redémarrage, heuristique de polarité, *etc.*) sur chacun des solveurs. Contrairement à la partie coopérative, la manière dont les solveurs entrent en compétition est choisie de manière statique au début de la recherche. Ce type d'approche ne permet ni d'identifier ni de traiter le cas où deux solveurs effectuent le même travail. En effet, même si deux solveurs ont des stratégies différentes, il n'est pas sûr qu'ils entrent en compétition. Il peut même arriver des cas où l'opposé se produit, c'est-à-dire que les deux solveurs vont entrer en phase de coopération. Cette coopération se traduit par un déséquilibre entre les deux phases et donc le plus souvent par un travail redondant de la part d'un des deux solveurs.

Pour éviter ce genre de situation, et ainsi préserver un schéma de type compétition/coopération, une méthode permettant d'ajuster dynamiquement l'heuristique de choix de la polarité est proposée. Pour cela, une mesure permettant d'estimer la distance entre deux solveurs est d'abord introduite. Puis, afin d'étudier le comportement des solveurs vis-à-vis de notre mesure, nous avons effectué un ensemble d'expérimentations.

10.1.1 Distance entre deux solveurs

Afin de savoir si deux solveurs sont en train d'effectuer le même travail, nous définissons la notion d'intention comme étant l'interprétation complète obtenue à partir de l'heuristique de choix de polarité associée à un solveur.

Définition 10.1. *Soient Σ une formule* CNF *et S un solveur utilisant une heuristique de choix de polarité définie par la fonction f. L'intention est définie comme :*

$$\mathcal{F} = \{y \in \mathcal{L}_\Sigma \textit{ tel que } f(x) = y \textit{ et } x \in \mathcal{V}_\Sigma\}.$$

L'interprétation complète $\mathcal{F}$ peut d'une certaine manière être vue comme l'espace de recherche que le solveur souhaite atteindre. De cette manière, nous pouvons supposer que deux solveurs sont proches dans l'espace de recherche s'ils ont l'intention d'explorer le même espace. À partir de la notion d'intention, il est possible de définir une mesure permettant d'estimer la distance entre deux solveurs.

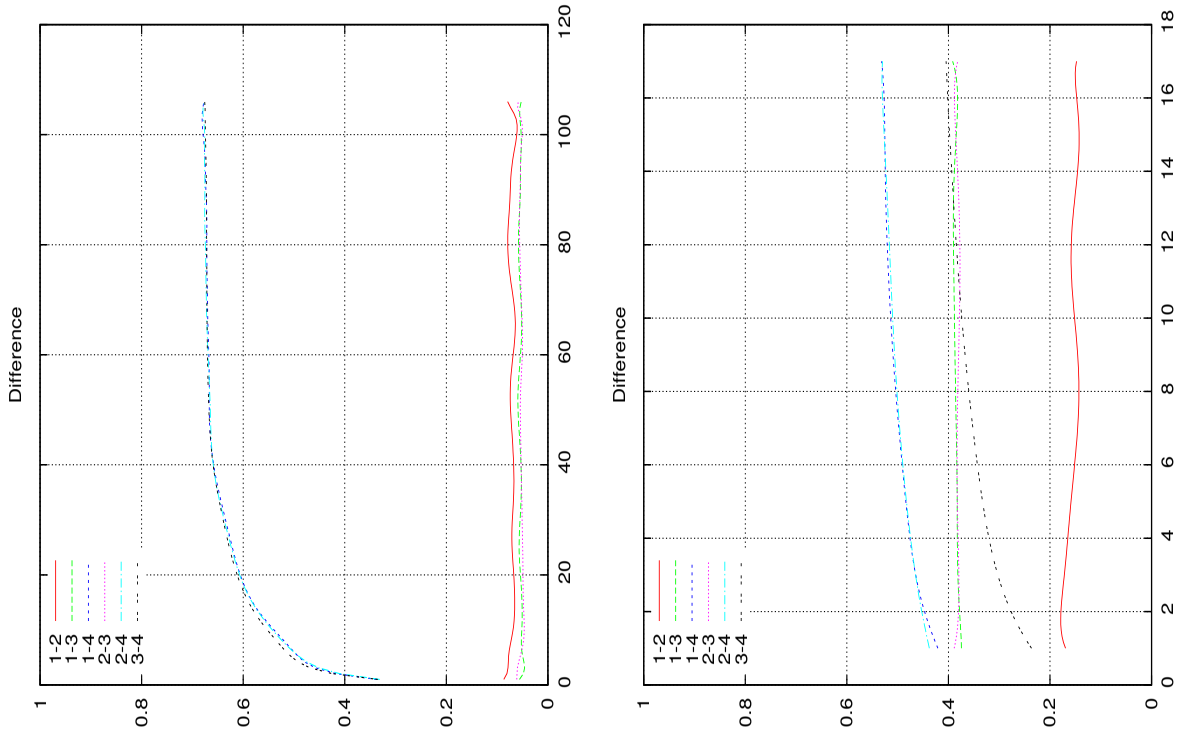
Définition 10.2. *Soient Σ une formule* CNF*, $(\mathcal{C}_i, \mathcal{F}_i)$ et $(\mathcal{C}_j, \mathcal{F}_j)$ deux solveurs $\mathcal{C}_i$ et $\mathcal{C}_j$ avec leurs intention respectives $\mathcal{F}_i$ et $\mathcal{F}_j$. La distance entre deux solveurs, notée κ, est alors définie comme :*

$$\kappa(\mathcal{C}_i, \mathcal{C}_j) = 1 - \frac{|\mathcal{F}_i \cap \mathcal{F}_j|}{|\mathcal{V}_\Sigma|}.$$

Remarquons que deux solveurs sont proches, d'après notre mesure, si leurs interprétations représentant leurs intentions respectives ont une faible distance de Hamming.

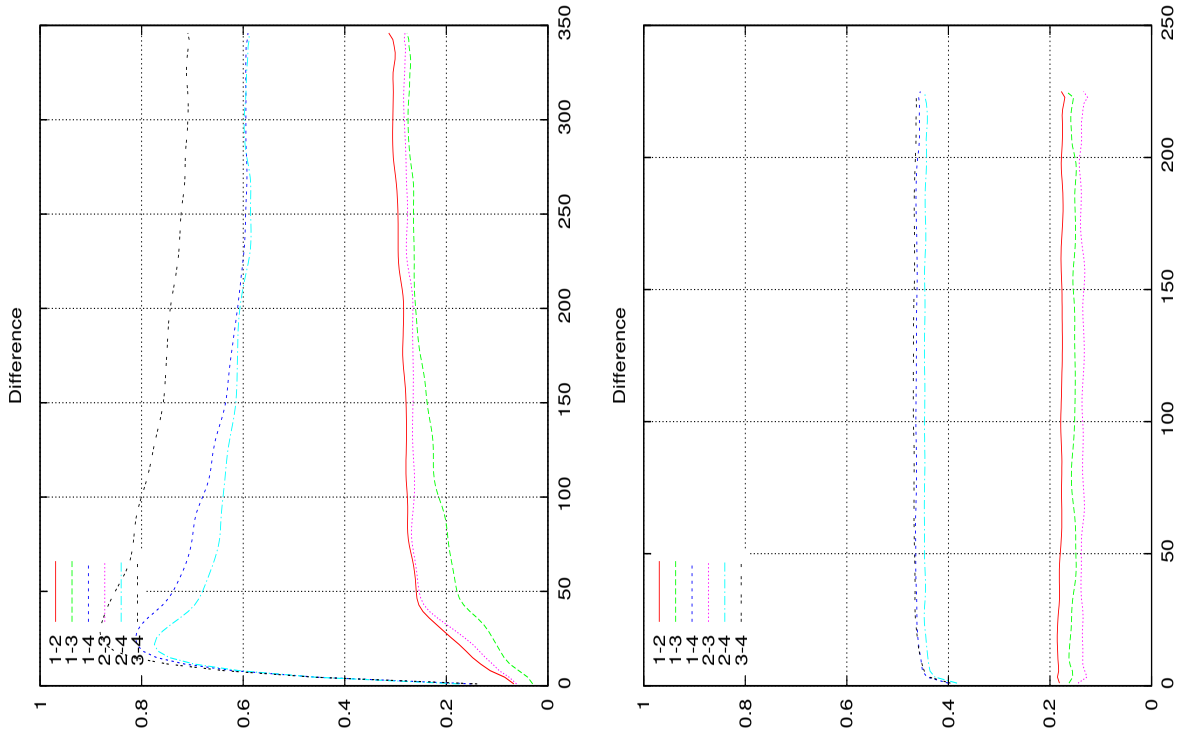
10.1.2 Évolution de la distance entre différents solveurs

Afin d'étudier l'évolution de la distance entre différents solveurs deux à deux, nous avons exécuté le solveur MANYSAT 1.1 (voir 5.3 pour la description) en récupérant la distance entre les différents solveurs tous les 5000 conflits. Les courbes de la figure 10.1 retranscrivent, de manière représentative, le comportement des différents solveurs les uns envers les autres.



(a) 9dlx_vliw_at_b_iq2

(b) schup-l2s-guid-1-k56



(c) vange-col-abb313GPIA-9-c

(d) countbitsarray08_32

FIGURE 10.1 – Comparaison deux à deux de la distance entre différents solveurs exécutés en parallèle. L'axe des abscisses représente le nombre de conflits atteint (divisé par 5000) et l'axe des ordonnées représente le résultat de notre mesure. Pour une meilleure lisibilité toutes les courbes ont été lissées.

Nous pouvons constater que les solveurs se divisent en deux catégories :

- la première catégorie regroupe les courbes où le cœur 4 n'apparaît pas, c'est-à-dire les courbes entre les cœurs 1-2, 1-3 et 2-3. Nous pouvons voir que les cœurs 1, 2 et 3 sont très proches les uns des autres. Ceci est facilement explicable pour les cœurs 1 et 2, étant donné qu'ils utilisent tous deux la même heuristique de choix de polarité (*progress saving*). En ce qui concerne le cœur 3, nous pensons que cela est dû au choix de la polarité initiale pour le *progress saving* (la phase étant affectée à « faux » au départ) ;
- la seconde catégorie regroupe les courbes où le cœur 4 apparaît. Nous observons que la distance entre le cœur 4 et n'importe quel autre cœur (courbes 1-4, 2-4, 3-4) est toujours supérieure au cas où le cœur 4 n'est pas présent dans le calcul de la distance (courbes 1-2, 1-3, 2-3). Ceci peut en partie être expliqué par le choix de la stratégie d'affectation de polarité du cœur 4. En effet, si nous considérons les unités de calcul 1 et 2, nous pouvons noter que lorsqu'ils apprennent une nouvelle clause α par analyse de conflit, elle est falsifiée par l'interprétation courante $\mathcal{P}$ (correspondant au *progress saving* du solveur, c'est-à-dire à son intention). Sans perte de généralité, supposons que le cœur 1 apprend la clause α. Cette clause est ensuite transférée au cœur 4 et le nombre d'occurrences des littéraux appartenant à α est incrémenté. Puisque l'heuristique de choix de polarité du cœur 4 est basée sur le nombre d'occurrences des littéraux, l'interprétation représentant son intention va avoir tendance à satisfaire les clauses proposées par les cœurs 1 et 2. Par conséquent, le solveur 4 s'éloigne du cœur 1 à chaque fois qu'il récupère une de ses clauses. En ce qui concerne le cœur 3, étant donné que son intention est proche de celle des cœurs 1 et 2, il est facile de comprendre pourquoi la distance qui le sépare du cœur 4 est grande.

10.1.3 Ajustement dynamique de la polarité

Après l'analyse du comportement des différents solveurs entre eux, nous avons choisi de définir une nouvelle heuristique de polarité. Puisque les solveurs peuvent être très proches dans l'espace de recherche nous avons décidé d'ajouter du « bruit » pour les éloigner les uns des autres. Pour cela, lorsque deux cœurs sont détectés comme « trop proches », nous choisissons d'inverser la polarité d'un des deux cœurs. Pour éviter que deux solveurs proches n'inversent en même temps leur polarité, l'ajustement se fait de manière unidirectionnelle, c'est-à-dire qu'un seul des deux solveurs ajuste sa polarité. De plus, afin de ne pas ralentir de manière excessive la vitesse du solveur, cet ajustement ne sera pas fait de manière systématique. La figure 10.2 décrit le transfert de messages et l'ajustement de la polarité entre deux solveurs. À chaque point de contrôle, représentés par des losanges, le cœur j demande au cœur i s'ils sont proches. Le cœur i lui répond par « oui » ou par « non » en fonction du résultat obtenu en mesurant sa distance avec le cœur j à l'aide de la mesure définie précédemment. Dans le cas où la réponse est négative, le cœur j utilise son heuristique de choix de polarité initiale. Dans le cas d'une réponse positive, le cœur j utilise son heuristique de polarité de manière inversée. Par exemple, supposons que le cœur b ait comme heuristique de polarité l'heuristique *false*. Dans ce cas, tant qu'un nouveau point de contrôle n'est pas atteint, le solveur affecte les prochains points de choix à vrai. Au prochain point de contrôle, l'heuristique de polarité peut de nouveau être réajustée. Ce processus est répété tant qu'une solution n'a pas été trouvée ou que le problème n'a pas été réfuté.

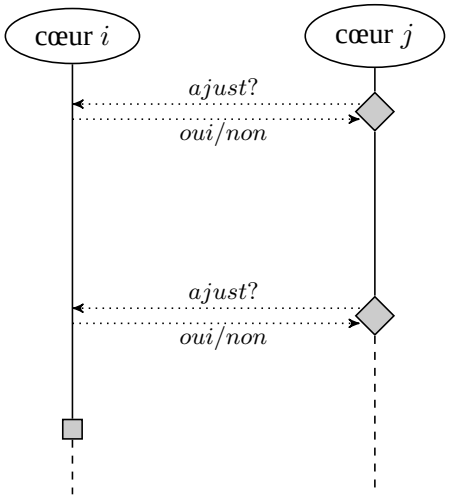


FIGURE 10.2 – Ajustement dynamique de la polarité d'un solveur vis-à-vis d'un autre.

10.2 Expérimentations

Afin d'étudier l'impact de cet ajustement, nous avons conduit les expérimentations suivantes. Dans un premier temps, nous avons choisi d'isoler l'heuristique de choix de polarité vis-à-vis des autres paramètres du solveur. Pour cela, hormis l'heuristique de choix de polarité, tous les cœurs utilisent des architectures identiques (politique de redémarrage, analyse de conflits et heuristique de choix de variables). Dans la seconde, nous avons intégré directement notre méthode au solveur MANYSAT 1.1. Enfin, nous étudions l'apport de notre schéma d'ajustement dans le solveur MANYSAT 1.5.

Avant d'entamer les expérimentations, il nous a fallu définir un schéma d'application pour notre méthode. En effet, dans la section précédente nous ne définissons pas formellement le cadre selon lequel notre ajustement doit être effectué.

Tout d'abord, à la vue des résultats obtenus concernant l'évaluation de la distance entre les solveurs pris deux à deux, notre méthode n'est pas appliquée sur le cœur 4 (cœur utilisant l'heuristique de polarité *occurence*). Ensuite, afin d'éviter des problèmes de cycle dans le protocole d'ajustement de polarité notre approche n'est pas non plus appliquée au cœur 1. La figure 10.3 schématise la manière selon laquelle l'ajustement des cœurs 2 et 3 est effectué. Nous avons choisi d'ajuster le cœur 2 et le cœur 3 dans le cas où ils sont proches du cœur 1, c'est-à-dire lorsque $\delta(\mathcal{C}_1, \mathcal{C}_2) \leq 0.1$ ou $\delta(\mathcal{C}_1, \mathcal{C}_3) \leq 0.1$. En ce qui concerne les points de contrôle, nous avons choisi de les effectuer tous les 5000 conflits. Cette valeur a été fixée expérimentalement.

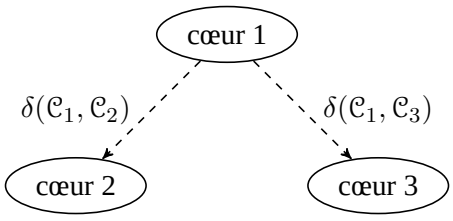


FIGURE 10.3 – Schéma d'ajustement MANYSAT 1.1.

L'ensemble des résultats expérimentaux reportés dans cette section ont été obtenus sur un Quad-core Intel XEON X5550 avec 32Gb de mémoire. Le temps CPU est limité à 900 secondes par unité de calcul. Pour ces expérimentations nous avons utilisé les 292 instances industrielles de la compétition SAT 2009 (Le Berre et Roussel 2009). Toutes les instances sont pré-traitées à l'aide de SATELITE (Eén et Biere 2005). Du fait du non déterminisme des solveurs parallèles, chaque solveur a été lancé trois fois sur toutes les instances. Les résultats reportés concerne l'exécution qui a résolu le plus d'instances (protocole utilisée lors de la compétition SAT de 2009).

10.2.1 Ajustement de l'heuristique de choix de polarité de MANYIDEM

Comme précisé précédemment, afin d'isoler l'heuristique de choix de polarité des autres composantes des solveurs SAT modernes, nous avons choisi d'utiliser un solveur, nommé MANYIDEM, avec une architecture identique pour l'ensemble des cœurs. Chaque unité de calculs utilise une politique de redémarrage définie par la suite de $luby(128)$, l'heuristique de choix de variable est VSIDS, le schéma d'apprentissage est le first-UIP et les clauses sont échangées si leur taille est inférieure à 8. En ce qui concerne l'heuristique de choix de polarité nous avons conservé celle implémentée dans la version initiale de MANYSAT 1.1 (voir tableau 5.1). MANYIDEM résout 72 instances satisfiables et 111 insatisfiables tandis que MANYIDEM avec notre technique d'ajustement résout 74 instances satisfiables et 120 insatisfiables. La figure 10.4 donne le nombre d'instances résolues (axe des abscisses) en fonction du temps (axe des ordonnées) par MANYIDEM avec et sans méthodes d'ajustement.

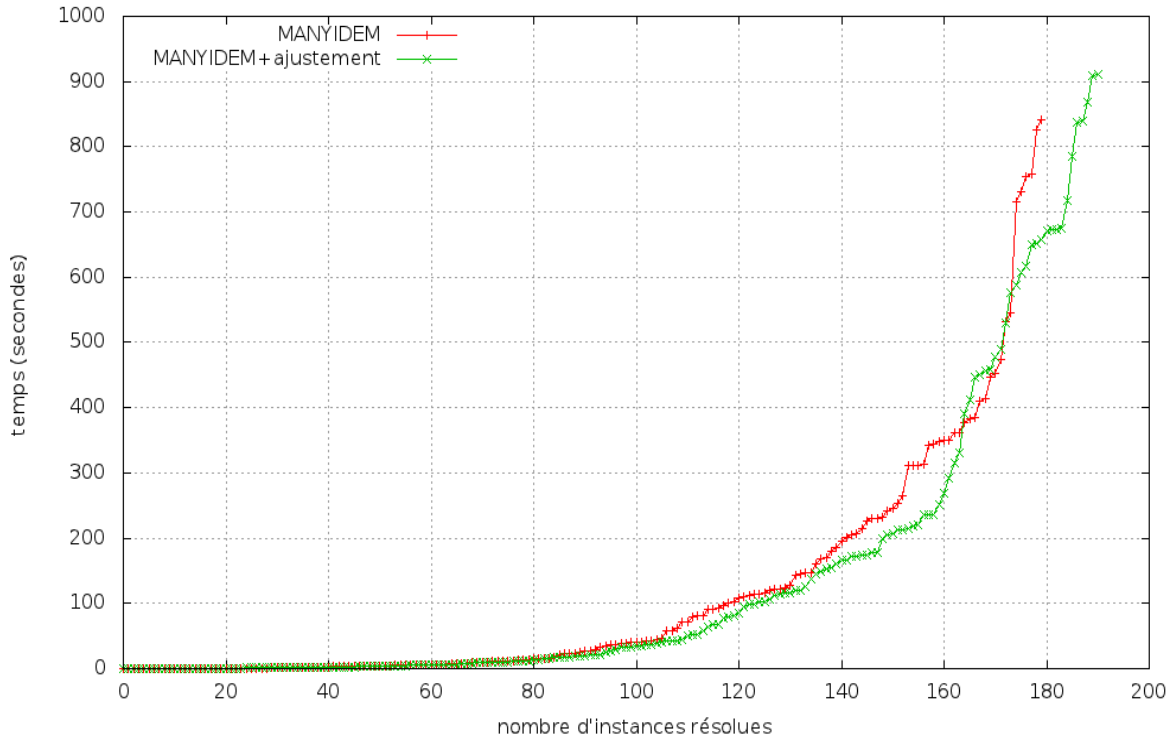


FIGURE 10.4 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour le solveur parallèle avec architecture identique intégrant ou pas notre technique d'ajustement d'heuristique de choix de polarité.

À la vue du nombre d'instances résolues (183 pour MANYIDEM et 194 pour MANYIDEM avec notre

technique d'ajustement), nous pouvons conclure que notre technique permet d'améliorer les performances du solveur. De plus, comme le montre le nuage de points de la figure 10.5, l'ajout de notre méthode permet toujours de résoudre de manière plus efficace les instances insatisfiables. Concernant les instances satisfiables nous pouvons noter que, même si les points sont cette fois-ci plus éparpillés, notre approche est plus compétitive.

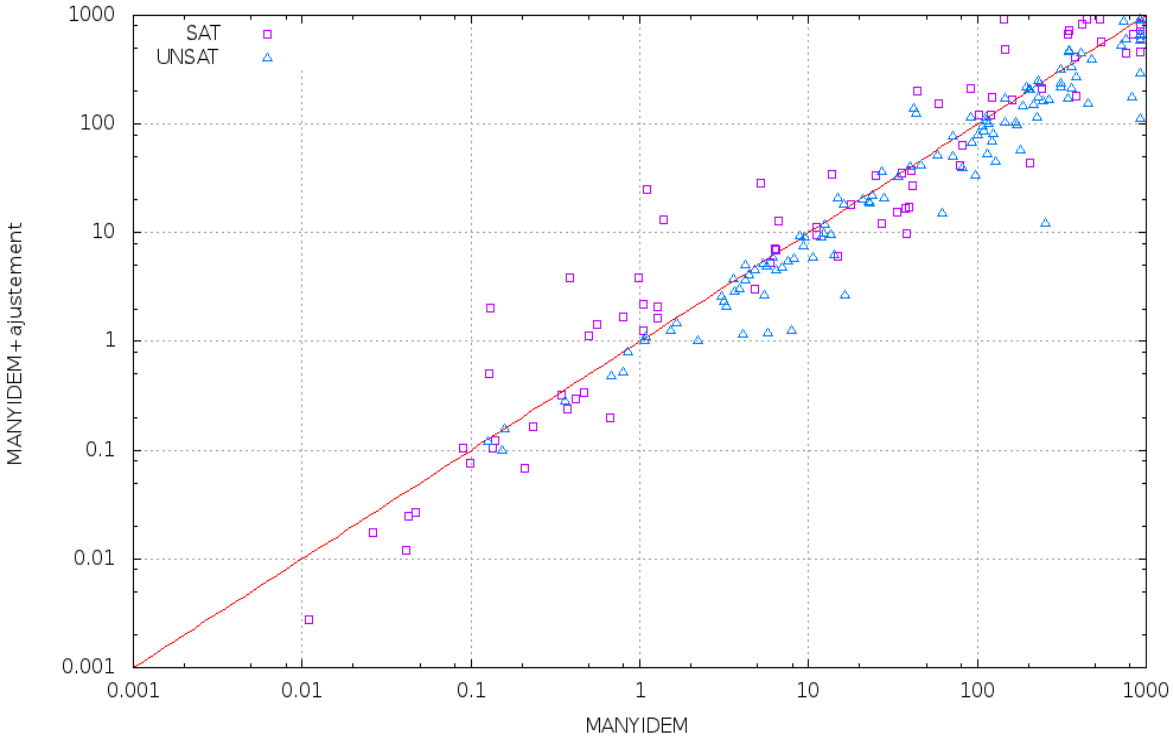


FIGURE 10.5 – Corrélation entre le temps mis par chacune des deux méthodes (MANYIDEM et MANYIDEM + ajustement) pour résoudre une instance donnée. Un point (temps MANYIDEM, temps MANYIDEM + ajustement) reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

10.2.2 Ajustement de l'heuristique de choix de polarité de MANYSAT 1.1

Dans cette partie, nous évaluons le gain apporté par notre technique d'ajustement de polarité dans le cadre du solveur MANYSAT 1.1 (Hamadi *et al.* 2009b). Ce dernier résout 73 instances satisfiables et 120 instances insatisfiables. Les figures 10.6 et 10.7 reportent les résultats obtenus par MANYSAT 1.1 utilisant ou non notre technique d'ajustement de la polarité. Ces dernières montrent clairement que l'ajout de notre méthode au sein du solveur SAT parallèle MANYSAT 1.1 permet d'améliorer sa compétitivité (81 instances satisfiables et 123 instances insatisfiables résolues). Si nous étudions plus finement les résultats à l'aide du nuage de points, nous nous rendons compte que comme dans le cas où les architectures sont identiques, notre méthode permet au solveur de résoudre plus efficacement les instances insatisfiables (les points ont tendance à se trouver en dessous de la diagonale). En ce qui concerne les instances satisfiables, les résultats expérimentaux montrent que notre approche permet de résoudre sensiblement plus d'instances (plusieurs points agglutinés sur la droite $x = 900$ signifiant que le solveur MANYSAT 1.1 sans notre technique n'a pas été capable de résoudre).

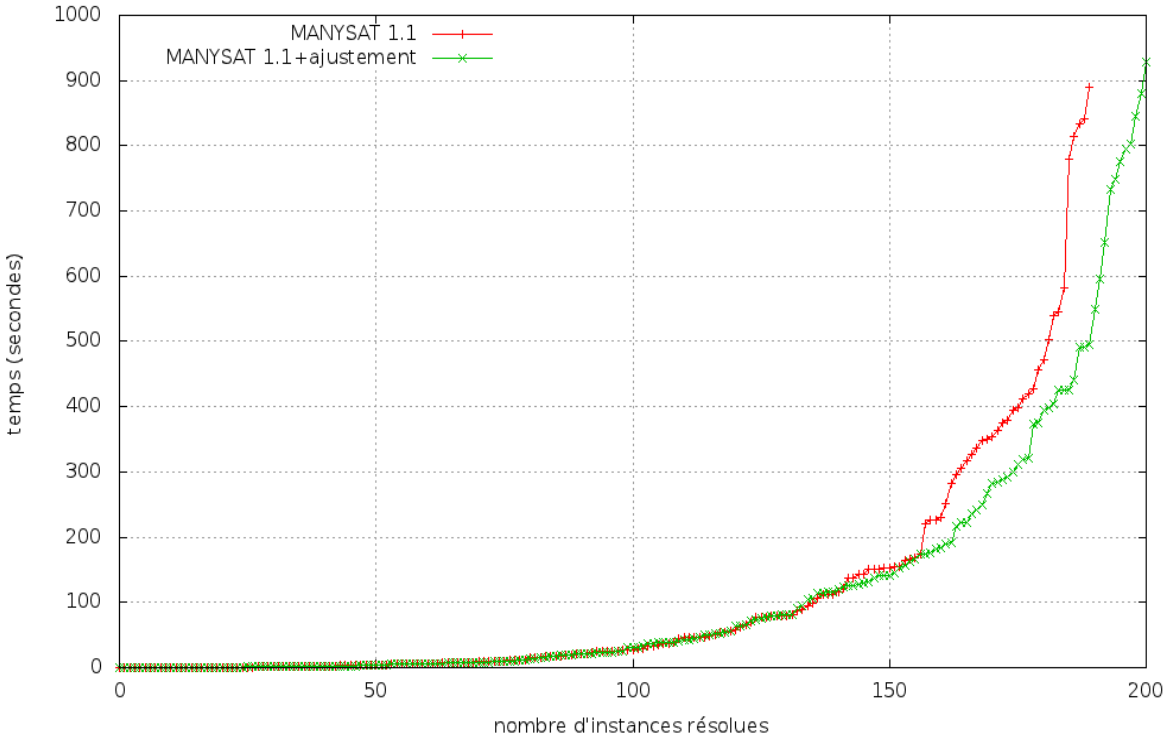


FIGURE 10.6 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour le solveur parallèle MANYSAT 1.1 intégrant ou pas notre technique d'ajustement de polarité.

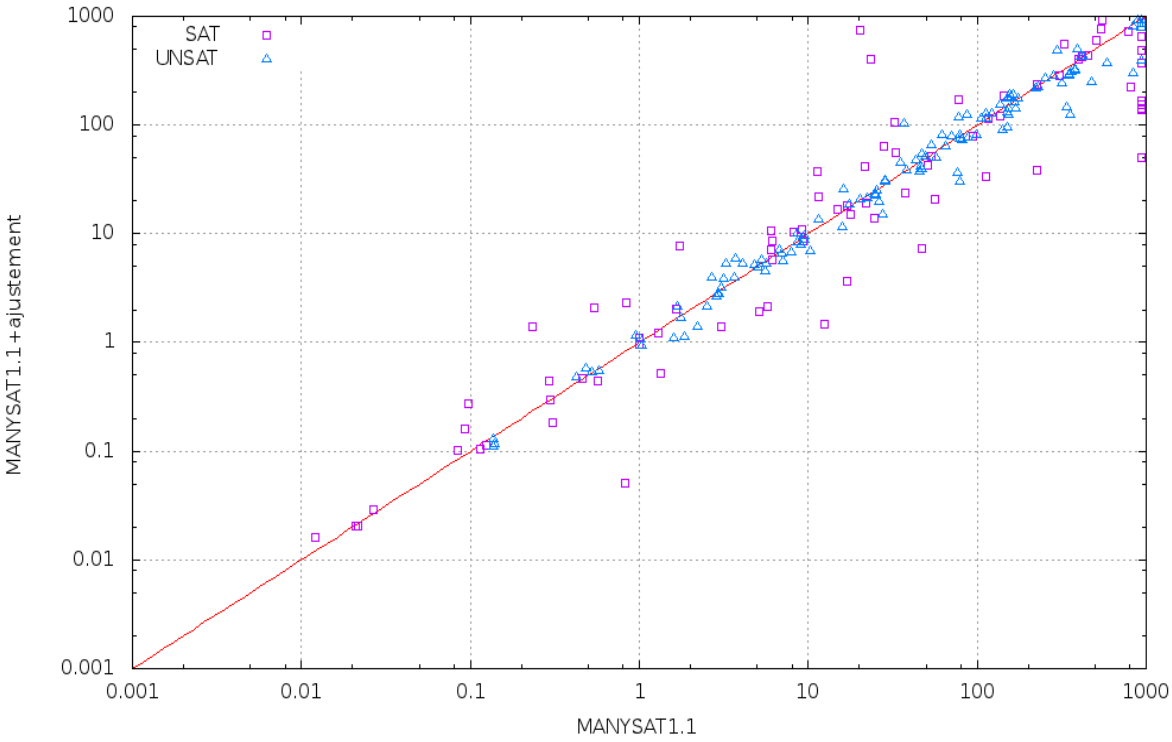


FIGURE 10.7 – Corrélation entre le temps mis par chacune des deux méthodes (MANYSAT 1.1 et MANYSAT 1.1 + ajustement) pour résoudre une instance donnée. Un point reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

10.2.3 Ajustement de l'heuristique de choix de polarité de MANYSAT 1.5

MANYSAT 1.5 (Guo *et al.* 2010) est une extension de MANYSAT 1.1 dans laquelle deux cœurs maîtres (cœur $\mathcal{M}1$ et cœur $\mathcal{M}2$) invoquent périodiquement deux esclaves (cœur $\mathcal{E}1$ et cœur $\mathcal{E}2$) dans le but d'intensifier une certaine partie de l'espace de recherche (voir 5.3.3). Ce solveur est basé sur le principe d'intensification/diversification. Afin d'évaluer la robustesse de notre approche nous avons incorporé notre stratégie d'ajustement de polarité au sein de MANYSAT 1.5. Néanmoins, puisque la structure du solveur est différente de celle de MANYSAT 1.1, nous avons défini une nouvelle topologie d'ajustement. Cette dernière, reportée sur la figure 10.8, consiste à ne considérer que les unités de calcul maître (c'est-à-dire $\mathcal{M}1$ et $\mathcal{M}2$) dans le processus d'ajustement. Nous avons choisi d'ajuster le cœur $\mathcal{M}2$ dans le cas où il est proche du cœur $\mathcal{M}1$ ($\delta(\mathcal{M}1, \mathcal{M}2) < 0.1$).

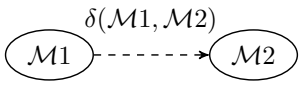


FIGURE 10.8 – Schéma d'ajustement de MANYSAT 1.5.

Les figures 10.9 et 10.10 illustrent les résultats obtenus par MANYSAT 1.5 (79 instances satisfiables et 124 instances insatisfiables résolues) et MANYSAT 1.5 avec notre technique d'ajustement (79 instances satisfiables et 127 instances insatisfiables résolues). Malgré le fait que l'ajout de notre méthode ne permet pas d'améliorer sensiblement les performances du solveur MANYSAT 1.5, nous pouvons observer qu'en ce qui concerne la résolution des instances insatisfiables la tendance précédemment observée se confirme. En effet, le nuage de point (figure 10.10) indique que la plupart des points représentant les instances insatisfiables se trouvent sous la diagonale.

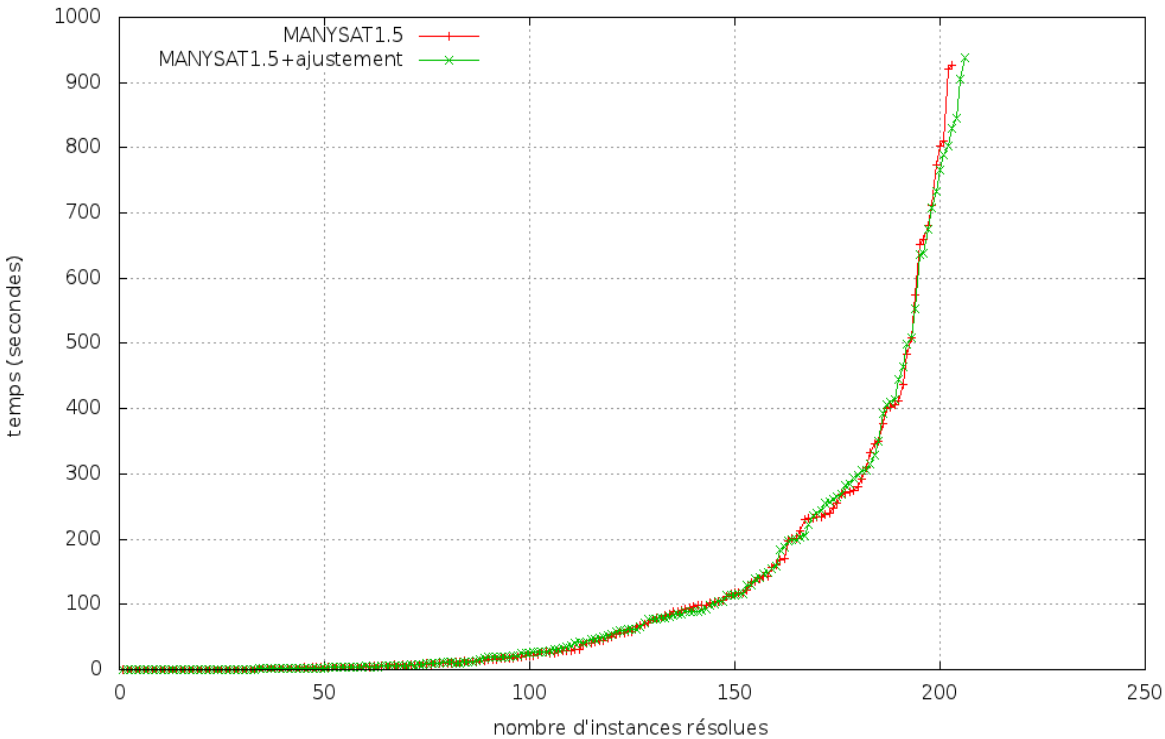


FIGURE 10.9 – Nombre d'instances résolues (abscisse) en fonction du temps (ordonnée) pour le solveur parallèle MANYSAT 1.5 intégrant ou pas notre technique d'ajustement de polarité.

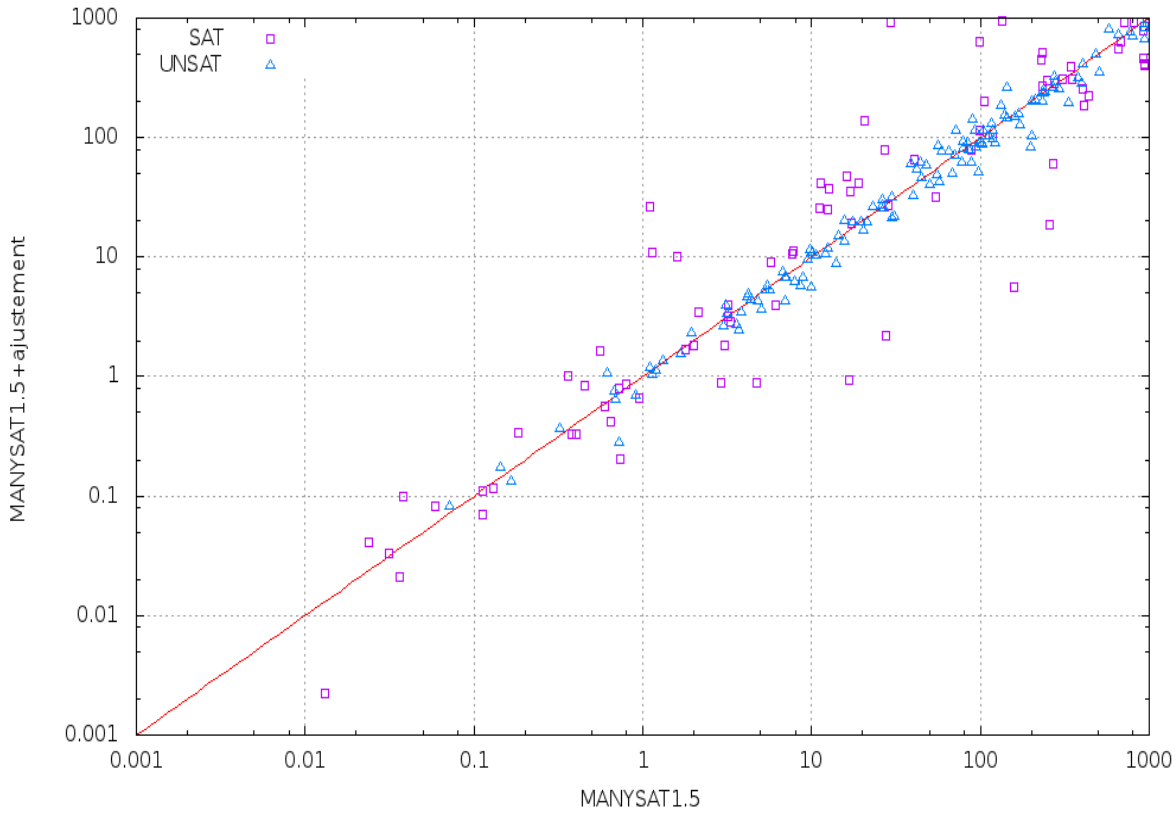


FIGURE 10.10 – Corrélation entre le temps mis par chacune des deux méthodes (MANYSAT 1.5 et MANYSAT 1.5 + ajustement) pour résoudre une instance donnée. Un point reporté sur le graphique correspond au résultat obtenu par chacun des solveurs sur une instance.

10.2.4 Résultats classés par famille d'instances

Les résultats expérimentaux reportés dans les sections précédentes ont permis de mettre en exergue le fait que l'ajout de notre stratégie d'ajustement dynamique de choix de polarité au sein de différentes versions du solveur MANYSAT permettait d'améliorer globalement leur efficacité. Dans cette partie, nous tentons d'étudier plus finement le comportement de notre approche en analysant les résultats obtenus sur certaines familles d'instances.

Les Tables 10.1 et 10.2 reportent les résultats obtenus sur certaines familles de problèmes sélectionnées parmi l'ensemble des instances industrielles de la compétition 2009. Les configurations de MANYSAT testées sont notées avec un « + » lorsque notre technique d'ajustement a été implémentée. Nous pouvons tout d'abord observer que notre technique d'ajustement permet d'améliorer les performances de MANYSAT sur plusieurs classes d'instances. De plus, il faut noter que, dans la plupart des cas, lorsque notre approche permet d'améliorer les performances d'une version de MANYSAT, elle améliore aussi les autres versions (vliw_unsat_2.0, q_query_3, AProVE, ...). Cette particularité permet de confirmer les résultats obtenus précédemment. En effet, dans le cadre de la résolution parallèle il est très difficile, à cause du non déterminisme, d'être catégorique lors de l'analyse des données expérimentales. Il est possible qu'avec de la « chance » un solveur soit considéré comme meilleur qu'un autre sans pour autant l'être réellement. Néanmoins, à la vue des résultats reportés dans cette section, nous pouvons affirmer que les gains observés dans les études expérimentales précédentes ne peuvent être imputées au

hasard.

	SAT	MANYIDEM	MANYIDEM +	MANYSAT1.1	MANYSAT1.1+	MANYSAT1.5	MANYSAT1.5+
velev-live-uns-2.0-ebuf	N	8.2	**5.8**	7.1	**5.6**	**4.3**	4.4
velev-pipe-uns-1.0-8	N	825.1	**177.4**	472	**249.2**	55.7	**48.8**
velev-pipe-o-uns-1.1-6	N	114.3	**52.5**	78.9	**30.3**	9.5	9.5
manol-pipe-g10bidw	N	28.2	**20.9**	26.3	**19.6**	21.4	**19.8**
manol-pipe-c10nidw_s	N	10.6	**6**	9.5	**8.9**	14	**8.9**
manol-pipe-f10ni	N	**204.5**	204.9	88.5	**77**	122	**90.9**
manol-pipe-f9b	N	92.8	**67.1**	**37.8**	38.6	68.1	**50.7**
goldb-heqc-i10mul	N	168.4	**102.6**	46.2	**41.2**	**26**	29.9
goldb-heqc-term1mul	N	22.9	**19.2**	22.2	**21.2**	8.6	**5.8**
goldb-heqc-dalumul	N	383.9	**268**	582.1	**374.2**	**47.6**	58.6
mizh-sha0-35-3	Y	**5.2**	28.4	227	**38.6**	157.2	**5.5**
mizh-sha0-36-4	Y	–	–	–	**138.1**	–	**844.8**
gus-md5-06	N	9.3	**7.6**	9	**7.9**	**10.1**	10.9
gus-md5-05	N	3.1	**2.6**	2.9	**2.6**	3.6	**2.7**
gus-md5-09	N	361.6	**330.4**	376	**318.5**	376.9	**316.2**
partial-10-13-s	Y	–	**672.9**	779.5	**733**	–	–
partial-10-15-s	Y	–	**458.3**	**326.5**	548.4	–	**414.9**
dated-5-17-u	N	244.7	**160.6**	154.2	**125.1**	**139.7**	155.4
dated-5-15-u	N	146	**102.9**	**87.4**	122.9	102.5	**88.1**
uts-l06-ipc5-h33-unknown	N	12.4	**9.8**	15.9	**11.5**	15.7	**13.4**
sortnet-8-ipc5-h19-sat	Y	–	**837**	–	**880.2**	–	–
cube-11-h13-unsat	N	121.7	**68.2**	142.3	**91.2**	201.8	**104.4**
vliw_unsat_2.0/9dlx_vliw_at_b_iq1	N	179.2	**57.4**	76.6	**36**	10	**5.5**
vliw_unsat_2.0/9dlx_vliw_at_b_iq2	N	453.1	**153.9**	335.8	**145.7**	30	**21.3**
vliw_unsat_2.0/9dlx_vliw_at_b_iq3	N	–	**651.7**	840.9	**299**	96.9	**51.6**
vliw_unsat_2.0/9dlx_vliw_at_b_iq4	N	–	**649.3**	–	**393.9**	170.2	**128.8**
vliw_unsat_2.0/9dlx_vliw_at_b_iq5	N	–	–	–	**928.5**	331.9	**196.5**
vliw_unsat_2.0/9dlx_vliw_at_b_iq6	N	–	–	–	–	508.3	**349.8**
vliw_unsat_2.0/9dlx_vliw_at_b_iq7	N	–	**674.4**	–	–	803.4	**707.9**

TABLE 10.1 – Résultats obtenus par différentes versions du solveur MANYSAT intégrant ou non la stratégie d'ajustement dynamique de l'heuristique de choix de polarité. Les temps reportés dans ce tableau sont exprimés en secondes.

	SAT	MANYIDEM	MANYIDEM +	MANYSAT1.1	MANYSAT1.1+	MANYSAT1.5	MANYSAT1.5+
AProVE07-16	N	101	**78.7**	78.4	**74.3**	**65.3**	77.3
AProVE07-27	N	716.1	**529.7**	349	**288.1**	**482.6**	498.2
AProVE09-06	Y	15	**6**	16.9	**3.7**	269.5	**60.2**
AProVE09-20	Y	27.1	**12.2**	24.7	**13.7**	**11.1**	26
vmpc_29	Y	–	–	–	**153.7**	927.2	**464**
vmpc_34	Y	–	–	–	**142.1**	–	**405.4**
minandmaxor032	N	5.4	**2.7**	2.9	**2.8**	3	**2.7**
minxorminand032	N	7.9	**1.3**	2.5	**2.2**	4.8	**4.3**
minxorminand064	N	253.4	**12.3**	24.9	**22.5**	98.1	**89.4**
minxor128	N	–	**112**	78.9	**74.7**	**144**	146.3
maxxororand032	N	–	**588.7**	168.5	**142.1**	**83.7**	89.8
minxorminand128	N	–	**291.7**	350.1	**291.2**	–	–
q_query_3_L200_coli.sat	N	185	**145.3**	150.4	**94.1**	**104.2**	114.5
q_query_3_l45_lambda	N	109.8	**85.6**	80.9	**72.3**	**92**	115.1
q_query_3_L80_coli.sat	N	127.9	**44.7**	**28.5**	30.9	89	**62.4**
q_query_3_l44_lambda	N	108.1	**94.8**	98.9	**80.9**	94.6	**83.1**
q_query_3_L150_coli.sat	N	214.2	**148.7**	**137.5**	156.1	118.3	**98.5**
BioInstances/rpoc_xits_07	N	313.1	**236.1**	150.3	**131.1**	399.5	**283.4**
UR-10-5p0	N	14.2	**6.3**	8.5	**8.3**	8.8	**6.7**
UR-10-5p1	Y	4.8	**3.1**	3.1	**1.4**	6.1	**4**
UTI-20-10p0	N	–	**909.7**	833.9	**802.8**	**272**	328.2
ACG-20-10p0	N	–	**656.9**	–	**845.7**	–	**830.3**
UCG-20-10p1	Y	841.8	**673.4**	–	**651**	652.4	**553.2**
ACG-20-10p1	Y	–	–	–	–	920.8	**789.8**
UTI-20-10p1	Y	–	**784.8**	–	–	681.7	**638**
cmu-bmc-longmult15	N	5.6	**4.9**	5	**4.9**	7	**6.8**
post-cbmc-aes-ee-r2-noholes	N	311.6	**235.5**	**112.5**	128.7	211.8	**205.5**
post-cbmc-aes-d-r2-noholes	N	362.1	**212.2**	**122.6**	126.8	234.7	**203.2**
post-c32s-gcdm16-22	Y	79.4	**41.9**	50.7	**43.3**	28.5	**26.9**

TABLE 10.2 – Résultats obtenus par différentes versions du solveur MANYSAT intégrant ou non la stratégie d'ajustement dynamique de l'heuristique de choix de polarité. Les temps reportés dans ce tableau sont exprimés en secondes.

10.3 Conclusion

Des travaux basés sur l'utilisation de l'heuristique de polarité pour régler certaines composantes des solveurs SAT modernes ont déjà été menés par le passé. Ainsi, dans (Biere 2008a), une politique de redémarrage basée sur l'heuristique de polarité *progress saving* est proposée. De notre coté, nous avons proposé d'utiliser cette notion pour contrôler la base de clauses apprises (voir le chapitre 9). Dans ce chapitre, nous avons présenté une nouvelle mesure basée sur ce principe permettant d'estimer la distance entre deux unités de calcul. Cette mesure a ensuite été utilisée dans le cadre d'une approche de type *portfolio* afin d'ajuster dynamiquement l'heuristique de polarité des différentes unités de calcul. Les résultats expérimentaux, menés sur l'ensemble des instances industrielles de la compétition SAT 2009, ont montré que notre méthode améliore sensiblement les performances du solveur MANYSAT 1.1. En ce qui concerne l'ajout de notre approche dans le solveur MANYSAT 1.5, nous ne pouvons pas conclure qu'il y a un réel gain. Néanmoins, à la vue des résultats (surtout sur les instances industrielles) il semble que notre approche soit très prometteuse.

Plusieurs perspectives s'ouvrent naturellement à la vue de ces résultats. Tout d'abord, nous envisageons d'étudier le comportement de notre approche sur les autres familles d'instances (instances aléatoires et académiques). De plus, il semble que la fréquence selon laquelle l'ajustement est effectué influe sur les performances. Il semble donc nécessaire d'étudier plus finement ce phénomène et de définir une stratégie dynamique pour régler ce paramètre. Nous pensons par la suite étudier d'autres critères d'ajustement. En effet, au lieu de se limiter à inverser l'heuristique de choix de polarité comme nous l'avons fait, une autre approche peut consister à en changer complètement pendant la phase de recherche. De cette manière un *portfolio* d'heuristiques de polarités est envisageable. Une autre perspective consiste à utiliser les informations fournies par la notion d'intention pour contrôler les flux de clauses apprises transitant entre les différents cœurs d'un solveur parallèle. Pour terminer, nous envisageons d'étudier notre approche sur d'autres solveurs de type *portfolio* (PLINGELING (Biere 2010), CRYPTOMINISAT// (Soos 2011), *etc.*) et plus particulièrement sur le solveur parallèle *portfolio* déterministe proposé par Hamadi *et al.* (2011). En effet, considérer une telle approche permet de simplifier fortement le protocole expérimental et surtout de supprimer le critère « chance » intrinsèque à l'utilisation de telles méthodes.

Conclusion

Nous avons présenté, dans cette partie, deux contributions s'appuyant sur l'heuristique de choix de polarité.

La première consiste en une nouvelle mesure, basée sur l'heuristique de choix de polarité *progress saving*, permettant d'évaluer la pertinence d'une clause. Nous avons montré expérimentalement, que dans le cadre d'une politique classique de réduction de la base de clauses apprises (voir section 3.1.5.2), utiliser notre mesure comme fonction de sélection permet d'obtenir une méthode aussi performante que celles proposées dans l'état de l'art. Une des particularités de la mesure proposée est qu'elle est dynamique et qu'elle peut être calculée sur des clauses ne participant pas à la recherche. Grâce à ces propriétés, nous avons pu proposer une nouvelle stratégie de réduction de la base de clauses apprises basée sur l'activation et la désactivation de clauses. Cette dernière, contrairement aux stratégies de réduction de l'état de l'art, ne supprime pas directement les clauses mais les met en sommeil lorsqu'elles sont jugées inutiles et les réactives lorsqu'elles sont jugées pertinentes. L'ensemble des expérimentations que nous avons conduit a montré que l'application de ce nouveau schéma dans un solveur CDCL permet d'améliorer sensiblement ses performances.

La seconde contribution s'inscrit quant à elle dans le cadre de la résolution de SAT en parallèle. Plus précisément, nous avons proposé une méthode permettant d'ajuster dynamiquement l'heuristique de choix de polarité des solveurs dans le cadre d'une approche de type *portfolio* (voir chapitre 5). Pour cela, nous nous sommes appuyés sur les heuristiques de choix de polarité de chaque solveur afin de déterminer si deux d'entre eux sont en train d'effectuer la même tâche. Lorsqu'une telle situation se produit, nous proposons simplement de modifier l'heuristique de choix de polarité d'un des deux solveurs. Les expérimentations condutes ont montré qu'appliquer notre méthode dans le solveur MANYSAT permet d'améliorer sensiblement ses performances.

Conclusion générale

Les travaux réalisés durant cette thèse apportent différentes contributions à la résolution des problèmes de satisfiabilité d'une formule propositionnelle (SAT) et de satisfaction de contraintes (CSP).

Après avoir introduit, dans la partie I, les différents éléments nécessaires à la compréhension de ce manuscrit, nous avons proposé, dans la seconde partie, différents algorithmes hybrides pour la résolution pratique des problèmes SAT et CSP.

- Tout d'abord, nous avons développé un algorithme hybride, appelé CDLS, dédié à la résolution de SAT. Il consiste à ajouter des clauses produites par résolutions à la formule afin de s'extraire des minima locaux. Pour cela, nous avons proposé d'adapter le concept d'analyse de conflits à partir d'un graphe d'implications au cadre de la recherche locale. Ce graphe, nommé « graphe conflit », est construit à partir d'une interprétation complète en considérant les clauses falsifiées comme les causes de l'échec (minimum local) et les clauses unisatisfaites comme les raisons ayant conduit à cette situation. Nous avons proposé deux approches pour analyser ce graphe. La première considère les notions de clauses critiques pour construire un chemin de clauses critiques, à partir duquel une nouvelle clause peut être produite par résolution. La seconde approche consiste à recontruire explicitement le graphe conflit à partir d'une interprétation partielle obtenue par propagation unitaire. Pour cela, nous construisons une nouvelle interprétation partielle à partir de l'interprétation complète courante issue de la recherche locale. Cette interprétation est ensuite utilisée afin de générer et d'analyser un graphe conflit.
- Les travaux effectués autour de CDLS nous ont conduit à proposer SATHYS, un démonstrateur complet hybride combinant recherche locale et solveur CDCL. Il s'appuie sur un nouveau schéma d'hybridation qui repose sur un flux bidirectionnel d'informations entre les différentes composantes mises en jeu. La particularité de ce schéma est que SATHYS peut être considéré comme une méthode de recherche locale utilisant un solveur CDCL afin de s'extirper des minima locaux, ou comme un solveur CDCL employant un solveur de recherche locale afin de se diriger vers les parties difficiles du problème. Ce solveur permet de résoudre plus d'instances que l'ensemble des solveurs de la littérature et offre pour la première fois une réponse à un challenge lancé à la communauté il y a maintenant presque 15 ans.
- Les excellentes performances obtenues par le solveur SATHYS nous ont conduit à proposer le solveur FAC-SOLVER pour résoudre le problème CSP, qui étend le schéma d'hybridation du solveur SATHYS et inclut le concept de variable FAC (*Falsified in All Constraints*). Afin d'obtenir une méthode hybride efficace, nous avons aussi proposé un solveur de recherche locale, nommé WCSP, intégrant certains des meilleurs critères d'échappement utilisés dans le cadre de SAT.

Nous avons, dans la troisième partie de ce manuscrit, proposé deux approches où l'heuristique de choix de polarité est utilisée de manière originale pour fournir des informations sur ce que pourrait être le futur espace de recherche exploré par un solveur SAT moderne (CDCL).

- La première consiste en une nouvelle mesure, basée sur l'heuristique de choix de polarité *progress saving*, permettant d'évaluer la pertinence d'une clause. Cette mesure est dynamique et peut être calculée sur des clauses ne participant pas à la recherche. Grâce à ses propriétés, nous avons proposé une nouvelle stratégie de réduction de la base de clauses apprises basée sur l'activation et la désactivation de clauses. Contrairement aux stratégies de réduction de l'état de l'art, ne supprime pas définitivement les clauses mais les place en sommeil lorsqu'elles sont jugées inutiles et les réactive lorsqu'elles sont jugées pertinentes. Ce travail a été récompensé par le prix de la meilleure contribution à la conférence SAT'2011.

– La seconde s'inscrit dans le cadre de la résolution de SAT en parallèle. Nous avons proposé une méthode permettant d'ajuster dynamiquement l'heuristique de choix de polarité des solveurs dans le cadre d'une approche de type *portfolio*. Nous nous sommes appuyés sur les heuristiques de choix de polarité de chaque solveur afin de déterminer si deux d'entre eux effectuent une tâche similaire. Lorsqu'une telle situation se produit, nous proposons simplement de modifier l'heuristique de choix de polarité d'un des deux solveurs. Nous espérons ainsi diversifier au mieux l'espace de recherche parcouru par les différents cœurs.

Ces différentes contributions ouvrent de nombreuses perspectives. Tout d'abord, nous envisageons de rendre l'algorithme CDLS complet. En effet, bien qu'il soit basé sur le même processus d'apprentissage que les solveurs SAT modernes (clause obtenue par résolution linéaire), il est impossible, même en considérant un temps infini, d'assurer qu'une réponse à la satisfiabilité ou à l'insatisfiabilité d'une formule sera fournie. En effet, contrairement à l'analyse de conflits effectuée dans les solveurs CDCL, il n'est pas garanti que la clause obtenue par notre processus d'analyse de conflits ne soit pas subsumée par une clause déjà présente dans la base. Afin de contourner ce problème, nous envisageons une approche permettant d'effectuer des traitements, *a postiori*, sur les clauses afin de s'assurer que celles-ci ne sont pas redondantes.

Les résultats obtenus par les solveurs SATHYS et FAC-SOLVER montrent l'intérêt du schéma d'hybridation proposée et laissent entrevoir de nombreuses perspectives. Une de ces perspectives concerne l'application de notre schéma d'hybridation à d'autres problèmes *NP-difficile*. En effet, nous avons pu voir aux travers des différentes études expérimentales que l'application de ce schéma au deux problèmes *NP-difficile* que ce sont SAT et CSP, conduit à produire systématiquement un solveur très robuste. Par conséquent, nous envisageons de travailler sur d'autres problèmes de cette classe, dans l'objectif de mesurer l'efficacité de ce schéma. Une autre piste que nous souhaitons explorer concerne l'adaptation de notre méthode hybride dans le cadre de la résolution parallèle. Nous étudions à l'heure actuelle une approche de type *portfolio* basé sur les mêmes principes que SATHYS et FAC-SOLVER.

En ce qui concerne le schéma de nettoyage de la base de clauses apprises que nous avons proposé, nous pensons que celui-ci peut encore être amélioré. En effet, beaucoup de paramètres contrôlant le cycle de vie d'une clause ont été définis de manière statique. Par conséquent, il pourrait être intéressant de définir une approche permettant de régler ces derniers de manière dynamique.

Enfin, nous avons pu voir que dans le cadre de la résolution du problème SAT à l'aide d'approches de type *portfolio*, l'heuristique de choix de polarité utilisée par chacun des solveurs permettait de fournir des informations intéressantes concernant l'espace de recherche qu'ils ont l'intention d'explorer. Il pourrait être intéressant d'utiliser de telles informations afin de contrôler plus finement l'échange de clauses apprises entre les différentes unités de calcul.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W

Bibliographie

Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02), volume 2470 de *Lecture Notes in Computer Science*, septembre 2002. Springer.

Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03), volume 2929 de *Lecture Notes in Computer Science*, Santa Margherita Ligure, Italy, mai 2003. Published in 2004. Springer.

Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04), volume 3542 de *Lecture Notes in Computer Science*, Vancouver (BC) Canada, mai 10-13 2004. Revised selected papers published in 2005. Sringer.

Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07), Hyderabad, India, janvier 6-16 2007.

Dimitris ACHLIOPTAS : *Random Satisfiability*, chapitre 8, pages 245–270. Volume 185 de , Biere *et al.* (2009), février 2009. ISBN 978-1-58603-929-5.

Sheldon B. AKERS : Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978. ISSN 0018–9340.

Marlene ARANGÚ, Miguel A. SALIDO et Federico BARBER : Ac2001-op : an arc-consistency algorithm for constraint satisfaction problems. *In Proceedings of the 23rd international conference on Industrial engineering and other applications of applied intelligent systems - Volume Part III*, IEA/AIE'10, pages 219–228, Berlin, Heidelberg, 2010. Springer-Verlag.

Josep ARGELICH, Daniel LE BERRE, Inês LYNCE, João P. MARQUES SILVA et Pascal RAPICAULT : Solving linux upgradeability problems using boolean optimization. *In* Inês LYNCE et Ralf TREINEN, éditeurs : *Proceedings First International Workshop on Logics for Component Configuration*, volume 29, pages 11–22, 2010.

Gilles AUDEMARD, Lucas BORDEAUX, Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : A generalized framework for conflict analysis. *In Proceedings of the 11th international conference on Theory and applications of satisfiability testing*, SAT'08, pages 21–27, Berlin, Heidelberg, 2008. Springer-Verlag.

Gilles AUDEMARD, Jean-Marie LAGNIEZ et Bertrand MAZURE : Approche hybride pour sat. *In 17eme congrés francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle(RFIA'10)*, pages 279–286, janvier 2010a.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : Analyse de conflits dans le cadre de la recherche locale. *In Journées Francophones de la Programmation par Contraintes(JFPC'09)*, jun 2009a.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : Integrating conflict driven clause learning to local search. *In Proceedings of International Workshop on Local Search Techniques in Constraint Satisfaction (affiliated to CP) (LSCS'09)*, Lisbon, Portugal, septembre 2009b. Electronic proceedings.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : Learning in local search. *In Proceedings of the 21st IEEE International Conference on Tools with Artificial Intelligence (ICTAI'09)*, pages 417–424, Newark, New Jersey, USA, novembre 2009c. IEEE Computer Press.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : Sathys : Sat hybrid solver. Solver description, sat-race 2010, 2010b.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : On freezeing and reactivating learnt clauses. *In Fourteenth International Conference on Theory and Applications of Satisfiability Testing(SAT'11)*, jun 2011a.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : Vers une gestion fine et dynamique de la base de clauses apprises. *In Septièmes Journées Francophones de Programmation par Contraintes (JFPC11)*, JFPC11, pages 15–24, juin 2011b.

Gilles AUDEMARD, Jean-Marie LAGNIEZ, Bertrand MAZURE et Lakhdar SAÏS : Boosting local search thanks to cdcl. *In* Christian FERMÜLLER et Andrei VORONKOV, éditeurs : *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6397 de *Lecture Notes in Computer Science*, pages 474–488. Springer Berlin / Heidelberg, 2010c.

Gilles AUDEMARD et Laurent SIMON : Glucose : a solver that predicts learnt clauses quality. Rapport technique, 2009a. SAT 2009 Competition Event Booklet, http//www.cril.univ-artois.fr/SAT09/solvers/booklet.pdf.

Gilles AUDEMARD et Laurent SIMON : Predicting learnt clauses quality in modern sat solver. *In Twenty-first International Joint Conference on Artificial Intelligence(IJCAI'09)*, pages 399–404, jul 2009b.

Fahiem BACCHUS : Enhancing davis putnam with extended binary clause reasoning. *In National Conference on Artificial Intelligence*, pages 613–619, 2002.

Fahiem BACCHUS et Paul van RUN : Dynamic variable ordering in csps. *In Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pages 258–275, London, UK, 1995. Springer-Verlag.

Rolf BACKOFEN et Sebastian WILL : A constraint-based approach to structure prediction for simplified protein models that outperforms other existing methods. *In* Catuscia PALAMIDESSI, éditeur : *Logic Programming*, volume 2916 de *Lecture Notes in Computer Science*, pages 49–71. Springer Berlin / Heidelberg, 2003.

Adrian BALINT, Michael HENN et Olivier GABLESKE : A novel approach to combine a sls- and a dpll-solver for the satisfiability problem. *In Proceedings of the Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 de *Lecture Notes in Computer Science*, pages 248–297, Swansea, Wales, United Kingdom, 2009. Springer.

Roman BARTÁK et Radek ERBEN : A new algorithm for singleton arc consistency. *In Proceedings FLAIRS*, volume 4, Miami Beach Floride, 2004. AAAI press.

Peter BARTH : A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization. Rapport technique, 1995.

Roberto J. BAYARDO JR. et Robert C. SCHRAG : Using csp look-back techniques to solve real-world sat instances. *In Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence (Rhode Island, USA), juillet 1997.

Paul BEAME, Henry A. KAUTZ et Ashish SABHARWAL : Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

Christopher J. BECK, Patrick PROSSER et Richard WALLACE : Variable ordering heuristics show promise. *In* Mark WALLACE, éditeur : *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 de *Lecture Notes in Computer Science*, pages 711–715. Springer Berlin / Heidelberg, 2004.

Nicolas BELDICEANU, Mats CARLSSON et Jean-Xavier RAMPON : Global constraint catalog. 2005.

Salem BENFERHAT, Jonathan BEN-NAIM, Robert JEANSOULIN, Mahat KHELFALLAH, Sylvain LAGRUE, Odile PAPINI, Nic WILSON et Éric WÜRBEL : Belief revision of gis systems : the results of revigis. *In* Lluis Godo (ED.), éditeur : *8th European Conferences on Symbolic and Quantitative Approaches to Reasoning with Uncertainty(ECSQARU'05)*, pages 452–464. Springer, jul 2005. Lecture Notes in Computer Science, Vol. 3571.

Philippe BESNARD, Éric GRÉGOIRE, Cédric PIETTE et Badran RADDAOUI : Mus-based generation of arguments and counter-arguments. *In 11th IEEE International Conference on Information Reuse and Integration(IRI'10)*, pages 239–244, Las Vegas (USA), aug 2010.

Christian BESSIÈRE et Marie-Odile CORDIER : Arc-consistency and arc-consistency again. *In Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, pages 108–113. AAAI Press, 1993.

Christian BESSIERE et Romuald DEBRUYNE : Optimal and suboptimal singleton arc consistency algorithms. *In Proceedings of the 19th international joint conference on Artificial intelligence*, pages 54–59, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

Christian BESSIÈRE, Eugene C. FREUDER et Jean-Charles RÉGIN : Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, January 1999. ISSN 0004-3702.

Christian BESSIÈRE, Jean-Charles RÉGIN, Roland H. C. YAP et Yuanlin ZHANG : An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165:165–185, July 2005. ISSN 0004-3702.

Christian BESSIÈRE, Pedro MESEGUER, Eugene FREUDER et Javier LARROSA : On forward checking for non-binary constraint satisfaction. *In* Joxan JAFFAR, éditeur : *Principles and Practice of Constraint Programming – CP'99*, volume 1713 de *Lecture Notes in Computer Science*, pages 88–102. Springer Berlin / Heidelberg, 1999.

Christian BESSIÈRE et Jean-Charles RÉGIN : Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. *In* Eugene FREUDER, éditeur : *Principles and Practice of Constraint Programming - CP96*, volume 1118 de *Lecture Notes in Computer Science*, pages 61–75. Springer Berlin / Heidelberg, 1996.

Armin BIERE : Adaptive restart strategies for conflict driven sat solvers. *In* Hans KLEINE BÜNING et Xishun ZHAO, éditeurs : *Theory and Applications of Satisfiability Testing - SAT 2008*, volume 4996 de *Lecture Notes in Computer Science*, pages 28–33. Springer Berlin / Heidelberg, 2008a.

Armin BIERE : Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4 (75-97):45, 2008b.

Armin BIERE : Precosat system description. SAT Competition, solver description, 2009.

Armin BIERE : Lingeling, plingeling, picosat and precosat at sat race 2010. Rapport technique, August 2010.

Armin BIERE, Alessandro CIMATTI, Edmund M. CLARKE, Masahiro FUJITA et Yunshan ZHU : Symbolic model checking using sat procedures instead of bdds. *In Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, DAC '99, pages 317–320, New York, NY, USA, 1999a. ACM.

Armin BIERE, Alessandro CIMATTI, Edmund M. Jr. CLARKE, Masahiro FUJITA et Yunshan ZHU : Symbolic model checking using sat procedures instead of bdds. *Design Automation Conference*, 0: 317–320, 1999b.

Armin BIERE, Marijn J. H. HEULE, Hans van MAAREN et Toby WALSH, éditeurs. *Handbook of Satisfiability*, volume 185 de *Frontiers in Artificial Intelligence and Applications*. IOS Press, février 2009. ISBN 978-1-58603-929-5.

Alain BILLIONNET et Alain SUTTER : An efficient algorithm fot the 3-satisfiability problem. *Operations Research Letters*, 12:29–36, 1992.

James R. BITNER et Edward M. REINGOLD : Backtrack programming techniques. *Communication ACM*, 18:651–656, November 1975.

Wolfgang BLOCHINGER, Carsten SINZ et Wolfgang KÜCHLIN : Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Comput.*, 29:969–994, July 2003. ISSN 0167-8191.

Avrim L. BLUM et Merrick L. FURST : Fast planning through planning graph analysis. *Artif. Intell.*, 90:281–300, February 1997. ISSN 0004-3702.

George BOOLE : *Les lois de la pensée*. Mathesis, 1854.

Frédéric BOUSSEMART, Frédéric HEMERY, Christophe LECOUTRE et Lakhdar SAIS : Boosting systematic search by weighting constraints. *In Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI'99)*, pages 482–486, Valence, Spain, août 2004.

Alfredo BRAUNSTEIN, Marc MÉZARD et Riccardo ZECCHINA : Survey propagation : An algorithm for satisfiability. *Random Struct. Algorithms*, 27:201–226, September 2005.

Laure BRISOUX, Éric GRÉGOIRE et Lakhdar SAÏS : Improving backtrack search for sat by means of redundancy. *In ISMIS '99 : Proceedings of the 11th International Symposium on Foundations of Intelligent Systems*, pages 301–309, London, UK, 1999. Springer-Verlag. ISBN 3-540-65965-X.

Randal E. BRYANT : Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992. ISSN 0360–0300.

Micheal BURO et Hans KLEINE-BÜNING : Report on the sat competition. Rapport technique, University of Paderborn, 1992.

Maria G. BUSCEMI et Ugo MONTANARI : A survey of constraint-based programming paradigms. *Computer Science Review*, 2(3):137 – 141, 2008.

Max BÖHM et Ewald SPECKENMEYER : A fast parallel sat-solver – efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381–400, 1996. ISSN 1012-2443.

Thierry CASTELL : Computation of Prime Implicates and Prime Implicants by a variant of the Davis and Putnam procedure. *In IEEE International Conference on Tools with Artificial Intelligence TAI'96, Toulouse France, 16/11/96-19/11/96*, pages 428–429, Los Alamitos, California, novembre 1996. IEEE Computer Society Press.

Byungki CHA et Kazuo IWAMA : Adding new clauses for faster local search. *In Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, AAAI'96, pages 332–337. AAAI Press, 1996.

Chin-Liang CHANG et Richard Char-Tung LEE : *Symbolic logic and mechanical theorem proving*. Academic Press, 1973.

Philippe CHATALIC et Laurent SIMON : Zres : The old davis-putman procedure meets zbdd. *In* David A. MCALLESTER, éditeur : *Automated Deduction - CADE-17, 17th International Conference on Automated Deduction, Pittsburgh, PA, USA, June 17-20, 2000, Proceedings*, volume 1831 de *Lecture Notes in Computer Science*, pages 449–454. Springer, 2000. ISBN 3-540-67664-3.

Hubie CHEN, Carla GOMES et Bart SELMAN : Formal models of heavy-tailed behavior in combinatorial search. *In* Toby WALSH, éditeur : *Principles and Practice of Constraint Programming - CP 2001*, volume 2239 de *Lecture Notes in Computer Science*, pages 408–421. Springer Berlin / Heidelberg, 2001.

Assef CHMEISS : Sur la consistance de chemin et ses formes partielles. *In Actes du Congrès AFCET-RFIA-96*, pages 212–219, Rennes, France, 1996.

Assef CHMEISS et Philippe JÉGOU : Path-consistency : when space misses time. *In Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, AAAI'96, pages 196–201. AAAI Press, 1996. ISBN 0-262-51091-X.

Assef CHMEISS, Ph. JÉGOU et L KEDDAR : On a generalization of triangulated graphs for domains decomposition of csps. *In IJCAI'03*, Acapulco (Mexique), aug 2003.

Assef CHMEISS et Philippe JÉGOU : Efficient path-consistency propagation. *Intelligence Journal for Artificial Intelligence Tools*, 7(2):121–142, 1998.

Wahid CHRABAKH et Rich WOLSKI : Gridsat : A chaff-based distributed sat solver for the grid. *In Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, SC '03, pages 37–, New York, NY, USA, 2003. ACM. ISBN 1-58113-695-1.

Geoffrey CHU et Peter J. STUCKEY : Pminisat : A parallelization of minisat 2.0. Solver description, sat-race 2008, 2008.

Stephen A. COOK : The complexity of theorem-proving procedures. *In STOC '71 : Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

Stephen A. COOK : A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, 8(4):28–32, 1976.

Martin C. COOPER : An optimal k-consistency algorithm. *Artificial Intelligence*, 41:89–95, November 1989.

James M. CRAWFORD : Solving satisfiability problems using a combination of systematic and local search. *In* DIMACS DIMACS. http://dimacs.rutgers.edu/Challenges/.

James M. CRAWFORD et Larry D. AUTON : Experimental results on the crossover point in satisfiability problems. *In Proceedings of the eleventh national conference on Artificial intelligence*, AAAI'93, pages 21–27. AAAI Press, 1993.

Nadia CREIGNOU, Sanjeev KHANNA et Madhu SUDAN : *Complexity classifications of boolean constraint satisfaction problems*, volume 7. Society for Industrial Mathematics, 2001.

CSP. Third international csp solver competition, 2008. http://cpai.ucc.ie/08/.

Mukesh DALAL : Efficient propositional constraint propagation. *In Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI'92)*, pages 409–414, San-Jose, California (USA), 1992.

Sylvain DARRAS, Gilles DEQUEN, Laure DEVENDEVILLE, Bertrand MAZURE, Richard OSTROWSKI et Lakhdar SAÏS : Using boolean constraint propagation for sub-clause deduction. *In Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 de *Lecture Notes in Computer Science*, pages 757–761, Sitges (Barcelona), Spain, octobre 2005. Springer.

Martin DAVIS, George LOGEMANN et Donald LOVELAND : A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5:394–397, 1962.

Martin DAVIS et Hilary PUTNAM : A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

Kenneth A. DE JONG et William M. SPEARS : Using genetic algorithms to solve np-complete problems. *In Proceedings of the third international conference on Genetic algorithms*, pages 124–132, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

Romuald DEBRUYNE et Christian BESSIÈRE : Some practicable filtering techniques for the constraint satisfaction problem. *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)* pro (2007), pages 412–417.

Rina DECHTER : Enhancement schemes for constraint processing : backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41:273–312, January 1990a.

Rina DECHTER : On the expressiveness of networks with hidden variables. *In Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, AAAI'90, pages 556–562. AAAI Press, 1990b. ISBN 0-262-51057-X.

Rina DECHTER et Itay MEIRI : Experimental evaluation of preprocessing techniques in constraint satisfaction problems. *In Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1*, pages 271–277, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

Rina DECHTER et Judea PEARL : Network-based heuristics for constraint-satisfaction problems. *Artificielle Intelligence*, 34:1–38, December 1987.

Rina DECHTER et Irina RISH : Directional resolution : the davis-putnam procedure revisited. *In* J. DOYLE, E. SANDEWALL et P. TORASSI, éditeurs : *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 134–145, 1994.

Louise DENNIS : An exploration of semantic resolution, 1994.

Gilles DEQUEN et Olivier DUBOIS : kcnfs : An efficient solver for random k-sat formulae. *In Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)* pro (2004), pages 486–501.

DIMACS. Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, octobre11-13 1993. http://dimacs.rutgers.edu/Challenges/.

Minh Binh DO et Subbarao KAMBHAMPATI : Planning as constraint satisfaction : solving the planning graph by compiling it into csp. *Artificial Intelligence*, 132:151–182, November 2001.

M. DORIGO et T. STÜTZLE : *Ant colony optimization*. the MIT Press, 2004.

Wiliam H. DOWLING et Jean H. GALLIER : Linear-time algorithms for testing satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.

Olivier DUBOIS, Pascal ANDRÉ, Yacine BOUFKHAD et Jacques CARLIER : Sat versus unsat. *In* D.S. JOHNSON et M.A. TRICK, éditeurs : *Selected papers of Second DIMACS Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University*, volume 26 de *DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society*, pages 415–436, 1996.

Olivier DUBOIS et Yacine BOUFKHAD : From very hard doubly balanced sat formulae to easy unbalanced sat formulae, variations of the satisfiability threshold. *In* J.G. DING-ZHU DU et P. PARDALOS, éditeurs : *Proceedings of the First Worshop on Satisifiability (SAT'96)*, Siena, Italy, mars 1996.

Derek L. EAGER, Edward D. LAZOWSKA et John ZAHORJAN : Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Softw. Eng.*, 12:662–675, May 1986. ISSN 0098-5589.

Niklas EÉN et Armin BIERE : Effective preprocessing in sat through variable and clause elimination. *In Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 de *Lecture Notes in Computer Science*, pages 61–75, St. Andrews, Scottland, juin 2005. Springer.

Niklas EÉN et Niklas SÖRENSON : An extensible sat-solver. *In Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)* pro (2004), pages 333–336.

Carlos EISENBERG et Boi FALTINGS : Using the breakout algorithm to identify hard and unsolvable subproblems. *In* Francesca ROSSI, éditeur : *Principles and Practice of Constraint Programming - CP 2003*, volume 2833 de *Lecture Notes in Computer Science*, pages 822–826. Springer Berlin / Heidelberg, 2003.

Thomas EITER et Georg GOTTLOB : On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 57(2-3):227–270, 1992. ISSN 0004-3702.

Thomas EITER et Georg GOTTLOB : Hypergraph transversal computation and related problems in logic and ai. *In JELIA '02 : Proceedings of the European Conference on Logics in Artificial Intelligence*, volume 2424 de *lncs*, pages 549–564, London, UK, 2002. Springer-Verlag. ISBN 3-540-44190-5.

Torsten FAHLE, Stefan SCHAMBERGER et Meinolf SELLMANN : Symmetry breaking. *In* Toby WALSH, éditeur : *Principles and Practice of Constraint Programming – CP 2001*, volume 2239 de *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2001.

Hai FANG et Wheeler RUML : Complete local search for propositional satisfiability. *In Proceedings of the Nineteenth American National Conference on Artificial Intelligence (AAAI'04)*, pages 161–166, 2004.

Lei FANG et Michael S. HSIAO : A new hybrid solution to boost sat solver performance. *In DATE '07 : Proceedings of the conference on Design, automation and test in Europe*, pages 1307–1313, San Jose, CA, USA, 2007. EDA Consortium. ISBN 978-3-9810801-2-4.

Brian FERRIS et Jon FROEHLICH : Walksat as an informed heuristic to dpll in sat solving. Rapport technique, CSE 573 : Artificial Intelligence, 2004.

Olivier FOURDRINOY, Éric GRÉGOIRE, Bertrand MAZURE et Lakhdar SAÏS : Exploring hybrid algorithms for sat. *In* G. SUTCLIFFER et A. Voronkov (EDS), éditeurs : *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'05)*, pages 33–37, Montego Bay, Jamaïque, dec 2005. short paper.

Jon William FREEMAN : *Improvements to Propositional Satisfiability Search Algorithms*. Ph.d. thesis, University of Pennsylvania, Department of Computer and Information Science, 1995.

Eugene C. FREUDER : Synthesizing constraint expressions. *Commun. ACM*, 21:958–966, November 1978. ISSN 0001-0782.

Eugene C. FREUDER : A sufficient condition for backtrack-bounded search. *J. ACM*, 32:755–761, October 1985.

Daniel FROST et Rina DECHTER : Look-ahead value ordering for constraint satisfaction problems. *In Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, pages 572–578, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8, 978-1-558-60363-9.

Alex FUKUNAGA : Efficient implementations of sat local search. *In Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)* pro (2005), pages 287–292.

Oliver GABLESKE et Marijn HEULE : Eagleup : Solving random 3-sat using sls with unit propagation. *In* Karem SAKALLAH et Laurent SIMON, éditeurs : *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695 de *Lecture Notes in Computer Science*, pages 367–368. Springer Berlin / Heidelberg, 2011.

Olivier GABLESKE et Julian RÜTH : Satun : A complete hybrid sat solver. *SAT2010 paper draft*, 2010.

Z. GALIL : On the complexity of regular resolution and the davis-putnam procedure. *Theoretical Computer Science*, 4:23–46, 1977.

Philippe GALINIER et Jin-Kao HAO : Tabu search for maximal constraint satisfaction problems. *In* Gert SMOLKA, éditeur : *Principles and Practice of Constraint Programming-CP97*, volume 1330 de *Lecture Notes in Computer Science*, pages 196–208. Springer Berlin / Heidelberg, 1997.

Philippe GALINIER et Jin-Kao HAO : A general approach for constraint solving by local search. *Journal of Mathematical Modelling and Algorithms*, 3:73–88, 2004.

John GASCHNIG : A constraint satisfaction method for inference making. *Proceedings of the Twelfth Annual Allerton Conference on Circuit and System Theory*, pages 268–277, 1974.

Martin GEBSER, Benjamin KAUFMANN, André NEUMANN et Torsten SCHAUB : Conflict-driven answer set solving. *In Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, pages 386–392, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

Pieter Andreas GEELEN : Dual viewpoint heuristics for binary constraint satisfaction problems. *In Proceedings of the 10th European conference on Artificial intelligence*, ECAI '92, pages 31–35, New York, NY, USA, 1992. John Wiley & Sons, Inc.

R. GÉNISSON et P. SIEGEL : A polynomial method for sub-clauses production. *In Proceedings of the sixth international conference on Artificial intelligence : methodology, systems, applications : methodology, systems, applications*, pages 25–34, River Edge, NJ, USA, 1994. World Scientific Publishing Co., Inc.

Richard GÉNISSON et Pierre SIEGEL : A polynomial method for sub-clauses production. *In Proceedings of Sixth International Conference on Artificila Intelligence : Methodology, Systems, Applications (AIMSA'94)*, pages 25–34, North Holland, 1994.

Ian GENT, Chris JEFFERSON et Ian MIGUEL : Watched literals for constraint propagation in minion. *In* Frédéric BENHAMOU, éditeur : *Principles and Practice of Constraint Programming - CP 2006*, volume 4204 de *Lecture Notes in Computer Science*, pages 182–197. Springer Berlin / Heidelberg, 2006.

Ian GENT, Ewan MACINTYRE, Patrick PRESSER, Barbara SMITH et Toby WALSH : An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. *In* Eugene FREUDER, éditeur : *Principles and Practice of Constraint Programming - CP96*, volume 1118 de *Lecture Notes in Computer Science*, pages 179–193. Springer Berlin / Heidelberg, 1996.

Ian P. GENT et Barbara SMITH : Symmetry breaking during search in constraint programming. *In Proceedings ECAI'2000*, pages 599–603, 1999.

Ian P. GENT et Tobby WALSH : Easy problems are sometimes hard. *Artificial Intelligence*, 70:335–345, 1994.

Matthew L. GINSBERG : Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

Matthew L. GINSBERG, Michael FRANK, Michael P. HALPIN et Mark C. TORRANCE : Search lessons learned from crossword puzzles. *In Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, AAAI'90, pages 210–215. AAAI Press, 1990. ISBN 0-262-51057-X.

Matthew L. GINSBERG et David A. MCALLESTER : Gsat and dynamic backtracking. *In* J. DOYLE, E. SANDEWALL et P. TORASSI, éditeurs : *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 226–236, 1994.

Fred W. GLOVER : Tabu search - part i. *ORSA Journal of Computing*, 1:190–206, 1989.

Fred W. GLOVER : Tabu search - part ii. *ORSA Journal of Computing*, 2:4–32, 1990.

Allen GOLDBERG : On the complexity of the satisfiability problem. Rapport technique, New York University, 1979.

Eugene GOLDBERG : Boundary points and resolution. *In* Oliver KULLMANN, éditeur : *SAT*, volume 5584 de *Lecture Notes in Computer Science*, pages 147–160. Springer, 2009. ISBN 978-3-642-02776-5.

Eugene P. GOLDBERG et Yakov NOVIKOV : Berkmin : a fast and robust sat-solver. *In In Proceedings of Design Automation and Test in Europe (DATE'02)*, pages 142–149, Paris, 2002.

Solomon W. GOLOMB et Leonard D. BAUMERT : Backtrack programming. *J. ACM*, 12:516–524, October 1965.

Carla P. GOMES, Bart SELMAN, Nuno CRATO et Henry KAUTZ : Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24:67–100, February 2000.

Carla P. GOMES, Bart SELMAN et Henry KAUTZ : Boosting combinatorial search through randomization. *In Proceedings of the Fifteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 431–437, Madison, Wisconsin, USA, juillet 1998. American Association for Artificial Intelligence Press. ISBN 0-262-51098-7.

Éric GRÉGOIRE, Bertrand MAZURE, Richard OSTROWSKI et Lakhdar SAÏS : Automatic extraction of functional dependencies. *In Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)* pro (2005), pages 122–132.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Extracting muses. *In Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'01)*, pages 387–391, Trento, Italy, aug 2006.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Local-search extraction of muses. *Constraints*, 12(3):325–344, septembre 2007.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Does this set of clauses overlap with at least one mus ? *In Proceedings of the 22nd International Conference on Automated Deduction (CADE-22*, pages 100–115, Montreal, P.Q., Canada, 2009a. Springer-Verlag (Berlin, Heidelberg). ISBN 978-3-642-02958-5.

Éric GRÉGOIRE, Bertrand MAZURE et Cédric PIETTE : Using local search to find msses and muses. *European Journal of Operational Research*, 199(3):640–646, décembre 2009b.

Éric GRÉGOIRE, Jean-Marie LAGNIEZ et Bertrand MAZURE : Un algorithme de résolution d'instances csp s'appuyant sur les variables fac. *In Septièmes Journées Francophones de Programmation par Contraintes (JFPC11)*, JFPC11, pages 145–153, juin 2011a.

Éric GRÉGOIRE, Jean-Marie LAGNIEZ et Bertrand MAZURE : A csp solver focusing on fac variables. *In* Jimmy LEE, éditeur : *Principles and Practice of Constraint Programming – CP 2011*, volume 6876 de *Lecture Notes in Computer Science*, pages 493–507. Springer Berlin / Heidelberg, 2011b.

Long GUO, Youssef HAMADI, Said JABBOUR et Lakhdar SAIS : Diversification and intensification in parallel sat solving. *In* David COHEN, éditeur : *Principles and Practice of Constraint Programming – CP 2010*, volume 6308 de *Lecture Notes in Computer Science*, pages 252–265. Springer Berlin / Heidelberg, 2010.

Long GUO et Jean-Marie LAGNIEZ : Ajustement dynamique de l'heuristique de polarité dans le cadre d'un solveur sat parallèle. *In Septièmes Journées Francophones de Programmation par Contraintes (JFPC11)*, JFPC11, pages 155–161, juin 2011a.

Long GUO et Jean-Marie LAGNIEZ : Dynamic polarity adjustment in a parallel sat solver (to appear). *In 23rd IEEE International Conference on Tools with Artificial Intelligence*, ICTAI'11, Nov. 7-9, 2011, Boca Raton, Florida, USA, novembre 2011b.

Richard GÉNISSON et Antoine RAUZY : Aspects algorithmiques des classes polynomiales du problème sat et des problèmes de satisfaction de constraintes. *In Rencontres Françaises en Intelligence Artificielle (RFIA'96)*, pages 96–108, Rennes (France), 1996.

Djamal HABET, Chu Min LI, Laure DEVENDEVILLE et Michel VASQUEZ : A hybrid approach for sat. *In Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)* pro (2002), pages 19–24.

Youssef HAMADI, Said JABBOUR, Cédric PIETTE et Lakhdar SAÏS : Concilier parallélisme et déterminisme dans la résolution de sat. *In Journées Francophones de la Programmation par Contraintes(JFPC'11)*, page A paraitre, Lyon, jun 2011.

Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : Learning for dynamic subsumption. *In Proceedings of the 2009 21st IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '09, pages 328–335, Washington, DC, USA, 2009a. IEEE Computer Society.

Youssef HAMADI, Said JABBOUR et Lakhdar SAIS : ManySAT : a Parallel SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2009b.

Youssef HAMADI, Saïd JABBOUR et Lakhdar SAÏS : Lysat : Solver description. SATRACE, solver description, 2010.

Ching-Chih HAN et Chia-Hoang LEE : Comments on mohr and henderson's path consistency algorithm. *Artificial Intelligence*, 36:125–130, August 1988.

Hyojung HAN et Fabio SOMENZI : On-the-fly clause improvement. *In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 209–222, Berlin, Heidelberg, 2009. Springer-Verlag.

Pierre HANSEN, Nenad MLADENOVIĆ et Dionisio PEREZ-BRITOS : Variable neighborhood decomposition search. *Journal of Heuristics*, 7:335–350, 2001.

Jin-Kao HAO et Dorne RAPHAËL : An empirical comparison of two evolutionary methods for satisfiability problems. *In International Conference on Evolutionary Computation (ICEC'94)*, pages 451–455, 1994.

Robert M. HARALICK et Gordon L. ELLIOTT : Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263 – 313, 1980.

William D. HARVEY : *Nonsystematic backtracking search*. Thèse de doctorat, 1995.

William S. HAVENS et Bistra N. DILKINA : A hybrid schema for systematic local search. *In Proceedings of The Seventeenth Canadian Conference on Artificial Intelligence (AI'2004)*, volume 3060 de *Lecture Notes in Computer Science*, pages 248–260. Springer, 2004. ISBN 978-3-540-22004-6.

Jacques HERBRAND, éditeur. *Écrits logiques*. Bibliotheque de philosophie contemporaine. Logique et philosophie des sciences. 1968.

Marijn HEULE, Matti JÄRVISALO et Armin BIERE : Clause elimination procedures for cnf formulas. *In Proceedings of the 17th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 357–371, Berlin, Heidelberg, 2010. Springer-Verlag.

Marijn J.H. HEULE et Hans van MAAREN : March_dl : Adding adaptive heuristics and a new branching strategy. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:47–59, mar 2006.

Edward A. HIRSCH et Arist KOJEVNIKOW : Unitwalk : A new sat solver that uses local serach guided by unit clause elimination. *Annals of Mathematics and Artificial Intelligence*, 43(1-4):91–111, 2005.

John H. HOLLAND : *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

John N. HOOKER et V. VINAY : Branching rules for satisfiability (extended abstract). *In Proceedings of the 14th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 426–437, London, UK, 1994. Springer-Verlag. ISBN 3-540-58715-2.

Holger H. HOOS : On the run-time behaviour of stochastic local search algorithms for sat. *In Proceedings of the Sixteenth American National Conference on Artificial Intelligence (AAAI'99)*, pages 661–666, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence. ISBN 0-262-51106-1.

Holger H. HOOS : An adaptive noise mechanism for walksat. *In Proceedings of the Eighteenth American National Conference on Artificial Intelligence (AAAI'02)*, pages 655–660, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0.

Holger H. HOOS et Thomas STÜTZLE : Local search algorithms for sat : An empirical evaluation. *Journal of Automated Reasoning*, 24:421–481, 2000.

Jinbo HUANG : The effect of restarts on the efficiency of clause learning. *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)* pro (2007), pages 2318–2323.

Tudor HULUBEI et Barry O'SULLIVAN : Search heuristics and heavy-tailed behaviour. *In* Peter van BEEK, éditeur : *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 de *Lecture Notes in Computer Science*, pages 328–342. Springer Berlin / Heidelberg, 2005.

Anthony HUNTER et Sébastien KONIECZNY : Measuring inconsistency through minimal inconsistent sets. *In Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 358–366, 2008.

Frank HUTTER, Dave TOMPKINS et Holger HOOS : Scaling and probabilistic smoothing : Efficient dynamic local search for sat. *In* Pascal VAN HENTENRYCK, éditeur : *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 de *Lecture Notes in Computer Science*, pages 241–249. Springer Berlin / Heidelberg, 2006.

Frank HUTTER, Dave A. D. TOMPKINS et Holger H. HOOS : Scaling and probabilistic smoothing : Efficient dynamic local search for SAT. *In Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)* pro (2002).

Joey HWANG et David MITCHELL : 2-way vs. d-way branching for csp. *In* Peter van BEEK, éditeur : *Principles and Practice of Constraint Programming - CP 2005*, volume 3709 de *Lecture Notes in Computer Science*, pages 343–357. Springer Berlin / Heidelberg, 2005.

Robert G. JEROSLOW et Jinchang WANG : Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

Narendra JUSSIEN et Olivier LHOMME : Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, juillet 2002.

Matti JÄRVISALO, Armin BIERE et Marijn HEULE : Blocked clause elimination. *In* Javier ESPARZA et Rupak MAJUMDAR, éditeurs : *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 de *Lecture Notes in Computer Science*, pages 129–144. Springer Berlin / Heidelberg, 2010.

Kalev KASK, Rina DECHTER et Vibhav GOGATE : Counting-based look-ahead schemes for constraint satisfaction. *In* Mark WALLACE, éditeur : *Principles and Practice of Constraint Programming - CP 2004*, volume 3258 de *Lecture Notes in Computer Science*, pages 317–331. Springer Berlin / Heidelberg, 2004.

Henri KAUTZ et Bart SELMAN : Ten challenges redux : Recent progress in propositional reasoning and search. *In Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, volume 2833 de *Lecture Notes in Computer Science*, pages 1–18, Kinsale, Ireland, septembre 2003. Springer.

Henry KAUTZ et Bart SELMAN : Walksat version 43. http://www.cs.washington.edu/homes-/kautz/walksat/walksat.

Henry KAUTZ et Bart SELMAN : Pushing the envelope : planning, propositional logic, and stochastic search. *In Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, AAAI'96, pages 1194–1201. AAAI Press, 1996.

Lukas KROC, Ashish SABHARWAL, Carla P. GOMES et Bart SELMAN : Integrating systematic and local search paradigms : a new strategy for maxsat. *In Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 544–551, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

Olivier KULLMANN : On a generalization of extended resolution. *Discrete Applied Mathematics*, 96-97:149 – 176, 1999.

François LABURTHE : Solving small tsps with constraints. *In Proceedings of the 14th International Conference on Logic Programming*, pages 316–330. MIT Press, 1997.

Jean-Louis LAURIERE : A Language and a Program for Stating and Solving Combinatorial Problems. *Artificial Intelligence*, 10:29–127, 1978.

D. LE BERRE et A. PARRAIN : Sat4j : Bringing the power of sat technology to the java platform. http ://www.sat4j.org.

Daniel LE BERRE : Exploiting the real power of unit propagation lookahead. *In Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, Boston University, Massachusetts, USA, June 14th-15th 2001. to appear.

Daniel LE BERRE et Pascal RAPICAULT : Dependency management for the eclipse ecosystem : eclipse p2, metadata and resolution. *In Proceedings of the 1st international workshop on Open component ecosystems*, pages 21–30, New York, NY, USA, 2009. ACM.

Daniel LE BERRE et Olivier ROUSSEL : « 12th international symposium on theory and applications of satisfiability testing ». May 2009.

Daniel LE BERRE, Olivier ROUSSEL et Laurent SIMON : presentation made at sat09 http://www.satcompetition.org/2009/sat09comp-slides.pdf, 2009.

Christophe LECOUTRE, Frederic BOUSSEMART et Fred HEMERY : Backjump-based techniques versus conflict-directed heuristics. *In Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '04, pages 549–557, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2236-X.

Christophe LECOUTRE, Frédéric BOUSSEMART et Fred HEMERY : Exploiting multidirectionality in coarse-grained arc consistency algorithms. *In* Francesca ROSSI, éditeur : *Principles and Practice of Constraint Programming – CP 2003*, Lecture Notes in Computer Science, pages 480–494. Springer Berlin / Heidelberg, 2003.

Christophe LECOUTRE et Stéphane CARDON : A greedy approach to establish singleton arc consistency. *In Proceedings of the 19th international joint conference on Artificial intelligence*, pages 199–204, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

Christophe LECOUTRE et Fred HEMERY : A study of residual supports in arc consistency. *In Proceedings of the 20th international joint conference on Artifical intelligence*, pages 125–130, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

Christophe LECOUTRE, Lakhdar SAÏS et Julien VION : Using sat encodings to derive csp value ordering heuristics. *Journal on Satisfiability, Boolean Modeling and Computation(JSAT)*, 1:169–186, may 2007.

Florian LETOMBE et João P. MARQUES-SILVA : Improvements to hybrid incremental sat algorithms. *In Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 de *Lecture Notes in Computer Science*, pages 161–181, Guangzhou, P. R. China, mai 2008. Springer. ISBN 978-3-540-79718-0.

Harry R. LEWIS : Renaming a set of clauses as horn set. *Journal of the Association for Computing Machinery*, 25:134–135, 1978.

Chu LI et Wen HUANG : Diversification and determinism in local search for satisfiability. 3569:158–172, 2005.

Chu Min LI et Anbulagan ANBULAGAN : Heuristics based on unit propagation for satisfiability problems. *In Proceedings of the 15th international joint conference on Artifical intelligence - Volume 1*, pages 366–371, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

Chu Min LI, Wanxia WEI et Harry ZHANG : Combining adaptive noise and look-ahead in local search for sat. *In Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 de *Lecture Notes in Computer Science*, pages 121–133, Lisbon, Portugal, 2007. Springer. ISBN 978-3-540-72787-3.

Paolo LIBERATORE : On the complexity of choosing the branching literal in dpll. *Artificial Intelligence*, 116:315–326, January 2000. ISSN 0004-3702.

Miron LIVNY et Myron MELMAN : Load balancing in homogeneous broadcast distributed systems. *SIGMETRICS Perform. Eval. Rev.*, 11:47–55, April 1982. ISSN 0163-5999.

Donalds W. LOVELAND : A linear format for resolution. 125:147–162, 1970.

Michael LUBY, Alistair SINCLAIR et David ZUCKERMAN : Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993. ISSN 0020-0190.

David LUCKHAM : Refinement theorems in resolution theory. 125:163–190, 1970.

Inês LYNCE et João MARQUES-SILVA : Probing-based preprocessing techniques for propositional satisfiability. *In Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI '03, pages 105–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-2038-3.

Alan K. MACKWORTH : Consistency in networks of relations. *Artificial Intelligence*, 8(1):99 – 118, 1977.

Alan K. MACKWORTH et Eugene C. FREUDER : The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65 – 74, 1985.

João P. MARQUES-SILVA et Karem A. SAKALLAH : Grasp—a new search algorithm for satisfiability. *In ICCAD '96 : Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7597-7.

Bertrand MAZURE, Lakhdar SAÏS et Éric GRÉGOIRE : Tabu search for SAT. *In Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 281–285, Providence (Rhode Island, USA), juillet 1997.

Bertrand MAZURE, Lakhdar SAÏS et Éric GRÉGOIRE : Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.

David A. MCALLESTER : An Outlook on Truth Maintenance. Rapport technique, MIT, 1980.

David A. MCALLESTER : Partial order backtracking. *Journal of Artificial Intelligence Research*, 1, 1993.

David A. MCALLESTER, Bart SELMAN et Henry A. KAUTZ : Evidence for invariants in local search. *In Proceedings of the Fourteenth American National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326, août 1997.

Amnon MEISELS, Solomon Eyal SHIMONY et Gadi SOLOTOREVSKY : Bayes networks for estimating the number of solutions of constraint networks. *Annals of Mathematics and Artificial Intelligence*, 28:169–186, January 2000.

Michel MINOUX : LTUR : A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Information Processing Letters*, 29(1):1–12, 15 septembre 1988.

Steven MINTON, Marc D. JOHNSTON, Andrew B. PHILIPS et Philip LAIRD : Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. *In Proceedings of the Eighth American National Conference on Artificial Intelligence (AAAI'90)*, 1990.

Steven MINTON, Mark D. JOHNSTON, Andrew B. PHILIPS et Philip LAIRD : Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, December 1992. ISSN 0004-3702.

David MITCHELL, Bart SELMAN et Hector J. LEVESQUE : Hard and easy distributions of sat problems. *In Proceedings of the Tenth American National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.

Roger MOHR et Thomas C. HENDERSON : Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225 – 233, 1986.

Roger MOHR et Gérald MASINI : Good old discrete relaxation. *In* Yves KODRATOFF, éditeur : *European Conference on Artificial Intelligence, ECAI'88*, pages 651–656, Munich, Allemagne, 1988. Pitman.

Remi MONASSON, Riccardo ZECCHINA, Scott KIRKPATRICK, Bart SELMAN et Lidror TROYANSKY : Determining computational complexity from characteristic'phase transitions'. *Nature*, 400 (6740):133–137, 1999.

Ugo MONTANARI : Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, 7:95 – 132, 1974.

Gordon E. MOORE : Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86 (1):82–85, 1998.

Paul MORRIS : The break out method for escaping from local minima. *In Proceedings of the Eleventh American National Conference on Artificial Intelligence (AAAI'93)*, pages 40–45, 1993.

Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK : Chaff : engineering an efficient sat solver. *In DAC '01 : Proceedings of the 38th annual Design Automation Conference* Moskewicz *et al.* (2001b), pages 530–535. ISBN 1-58113-297-2.

Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK : Chaff : engineering an efficient sat solver. *In DAC '01 : Proceedings of the 38th annual Design Automation Conference*, pages 530–535, New York, NY, USA, juin 2001b. ACM. ISBN 1-58113-297-2.

Malek MOUHOUB et Bahareh JAFARI : Heuristic techniques for variable and value ordering in csps. *In Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 457–464, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0557-0.

Alexander NADEL et Vadim RYVCHIN : Assignment stack shrinking. *In* Ofer STRICHMAN et Stefan SZEIDER, éditeurs : *Theory and Applications of Satisfiability Testing - SAT 2010*, volume 6175 de *Lecture Notes in Computer Science*, pages 375–381. Springer Berlin / Heidelberg, 2010.

Alexandre NADEL, Maon GORDON, Amit PATI et Ziad HANA : Eureka-2006 sat solver. SAT-Race, solver description, 2006.

Bernard A. NADEL : Some applications of the constraint satisfaction problem. *In Workshop on Constraint Directed Reasoning Working Notes, Boston, Mass.*, AAAI'90, 1990.

Alexander NAREYEK, Stephen F. SMITH et Christian M. OHLER : Local search for heuristic guidance in tree search. *In ecai04*, pages 1069–1070, Valencia (Espagne), aug 2004.

Koji NONOBE et Toshihide IBARAKI : A tabu search approach to the constraint satisfaction problem as a general problem solver. *European Journal of Operational Research*, 106(2-3):599 – 623, 1998.

Yakov NOVIKOV : Local search for boolean relations on the basis of unit propagation. *In Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10810–, Washington, DC, USA, 2003. IEEE Computer Society.

Richard OSTROWSKI, Éric GRÉGOIRE, Bertrand MAZURE et Lakhdar SAÏS : Recovering and exploiting structural knowledge from cnf formulas. *In Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP'02)* pro (2002), pages 185–199.

Peng Sing OW et Thomas E. MORTON : Filtered beam search in scheduling. *International Journal of Production Research*, 26:35–62, 1988.

Christos H. PAPADIMITRIOU : *Computational Complexity*. Addison Wesley Pub. Co., 1994.

Christos H. PAPADIMITRIOU et David WOLFE : The complexity of facets resolved. *J. Comput. Syst. Sci.*, 37(1):2–13, 1988. ISSN 0022-0000.

Lionel PARIS, Richard OSTROWSKI, Pierre SIEGEL et Lakhdar SAÏS : From horn strong backdoor sets to ordered strong backdoor sets. *In Proceedings of the artificial intelligence 6th Mexican international conference on Advances in artificial intelligence*, MICAI'07, pages 105–117, Berlin, Heidelberg, 2007. Springer-Verlag.

Andrew J. PARKES et Joachim P. WALSER : Tuning local search for satisfiability testing. *In Proceedings of the Thirteenth American National Conference on Artificial Intelligence (AAAI'96)*, pages 356–362, juillet 1996.

Gilles PESANT, Michel GENDREAU, Jean-Yves POTVIN et Jean-Marc ROUSSEAU : An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Science*, 32:12–29, January 1998. ISSN 1526-5447.

Duc PHAM, John THORNTON, Charles GRETTON et Abdul SATTAR : Advances in local search for satisfiability. *In* Mehmet ORGUN et John THORNTON, éditeurs : *AI 2007 : Advances in Artificial Intelligence*, volume 4830 de *Lecture Notes in Computer Science*, pages 213–222. Springer Berlin / Heidelberg, 2007a.

Duc-Nghia PHAM et Charles GRETTON : gnovelty+ (v. 2). *In SAT 2009 Competitive Events Booklet*, 2009. URL http://www.cs.bham.ac.uk/~grettonc/CharlesSAT09.pdf.

Duc Nghia PHAM, John THORNTON et Abdul SATTAR : Building structure into local search for sat. *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)* pro (2007), pages 2359–2364.

Cédric PIETTE, Youssef HAMADI et Saïs LAKHDAR : Vivifying propositional clausal formulae. *In Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'03)*, pages 525–529, Patras (Greece), juillet 2008.

Knot PIPATSRISAWAT et Adnan DARWICHE : A lightweight component caching scheme for satisfiability solvers. *In SAT*, pages 294–299, 2007.

Knot PIPATSRISAWAT et Adnan DARWICHE : On the power of clause-learning sat solvers with restarts. *In Proceedings of the 15th international conference on Principles and practice of constraint programming*, CP'09, pages 654–668, Berlin, Heidelberg, 2009a. Springer-Verlag.

Knot PIPATSRISAWAT et Adnan DARWICHE : Width-based restart policies for clause-learning satisfiability solvers. *In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 341–355, Berlin, Heidelberg, 2009b. Springer-Verlag. ISBN 978-3-642-02776-5.

Daniele PRETOLANI : *Satisfiability and Hypergraphs*. Thèse de doctorat, dipartimento di Informatica : Universita di Pisa, Genova, Italia, mars 1993. TD-12/93.

Daniele PRETOLANI : Efficiency and stability of hypergraph sat algorithms. *Cliques, coloring, and satisfiability : second DIMACS implementation challenge, October 11-13, 1993*, page 479, 1996.

Patrick PROSSER : Hybrid algorithms for the constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.

William V. QUINE : *Methods of logics*. Henry Holt, 1950.

Antoine RAUZY : Polynomial restrictions of sat : What can be done with an efficient implementation of the davis and putnam's procedure. *In* U. MONTANARI et F. ROSSI, éditeurs : *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP'95)*, volume 976 de *Lecture Notes in Computer Science*, pages 515–532, Cassis, France, septembre 1995. Springer.

Jean-Charles RÉGIN : The symmetric alldiff constraint. *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 420–425, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

John Alan ROBINSON : A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 12:23–41, 1965.

John Alan ROBINSON : Automatic deduction with hyperresolution. *In Automation of Reasoning-Classical Papers on Computational Logic*, volume 1 and 2. Springer, 1983.

Francesca ROSSI, Charles PETRIE et Vasant DHAR : On the equivalence of constraint satisfaction problems. *In Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, 1990.

Lawrence RYAN : *Efficient algorithms for clause-learning SAT solvers*. Thèse de doctorat, Simon Fraser University, 2004.

Daniel SABIN et Eugene FREUDER : Contradicting conventional wisdom in constraint satisfaction. *In* Alan BORNING, éditeur : *Principles and Practice of Constraint Programming*, volume 874 de *Lecture Notes in Computer Science*, pages 10–20. Springer Berlin / Heidelberg, 1994.

Thomas SCHIEX et Gérard VERFAILLIE : Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 3:48–55, 1993.

Dale SCHUURMANS, Finnegan SOUTHEY et Robert C. HOLTE : The exponentiated subgradient algorithm for heuristic boolean programming. *In Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1*, pages 334–341, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

Bart SELMAN et Henry A. KAUTZ : Domain-independent extensions to gsat : Solving large structured satisfiability problems. *In IJCAI*, pages 290–295, 1993.

Bart SELMAN, Henry A. KAUTZ et Bram COHEN : Local search strategies for satisfiability testing. *In Working notes of the DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*, 1993.

Bart SELMAN, Henry A. KAUTZ et Bram COHEN : Noise strategies for improving local search. *In Proceedings of the Twelfth American National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, 1994.

Bart SELMAN, Henry A. KAUTZ et David A. MCALLESTER : Ten challenges in propositional reasoning and search. *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 50–54, Nagoya, Japan, août 23-29 1997.

Claude Elwood SHANNON : A symbolic analysis of relay and switching circuits. Thesis (m.s.), Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1940.

Haiou SHEN et Hantao ZHANG : Another complete local search method for sat. *In* Geoff SUTCLIFFE et Andrei VORONKOV, éditeurs : *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 3835 de *Lecture Notes in Computer Science*, pages 595–605. Springer Berlin / Heidelberg, 2005.

Moninder SINGH : Path consistency revisited. *In Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, TAI '95, pages 318–, Washington, DC, USA, 1995. IEEE Computer Society.

Carsten SINZ, Wolfgang BLOCHINGER et Wolfgang KÜCHLIN : Pasat – parallel sat-checking with lemma exchange : Implementation and applications. *Electronic Notes in Discrete Mathematics*, 9:205 – 216, 2001. ISSN 1571-0653. LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001).

Barbara M. SMITH et Stuart A. GRANT : Trying harder to fail first. *In Thirteenth European Conference on Artificial Intelligence (ECAI 98)*, pages 249–253. John Wiley & Sons, 1997.

Mate SOOS : Cryptominisat. Rapport technique, available on WebSite : http://www.msoos.org, 2011.

Niklas SÖRENSSON et Armin BIERE : Minimizing learned clauses. *In Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pages 237–243, Berlin, Heidelberg, 2009. Springer-Verlag.

Niklas SÖRENSSON et Niklas EÉN : Minisat 2.1 and minisat++ 1.0 - sat race 2008 editions. *SAT 2009 competitive events booklet : preliminary version*, page 31, 2009.

Francis SOURD et Philippe CHRÉTIENNE : Fiber-to-object assignment heuristics. *European Journal of Operational Research*, 117(1):1–14, 1999.

Richard M. STALLMAN et Gerald J. SUSSMAN : Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135 – 196, 1977. ISSN 0004-3702.

Larry J. STOCKMEYER : The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1 – 22, 1976.

Sathiamoorthy SUBBARAYAN et Dhiraj PRADHAN : Niver : Non-increasing variable elimination resolution for preprocessing sat instances. *In* Holger HOOS et David MITCHELL, éditeurs : *Theory and Applications of Satisfiability Testing*, volume 3542 de *Lecture Notes in Computer Science*, pages 899–899. Springer Berlin / Heidelberg, 2005.

Sébastien TABARY : *Exploiter les conflits pour réduire l'effort de recherche en satisfaction de contraintes*. Thèse de doctorat, Université d'Artois, Lens, nov 2007.

Dave A. D. TOMPKINS : Ubcsat. http://www.cs.ubc.ca/~davet/ubcsat.

G. TSEITIN : On the complexity of proofs in propositional logics. 2, 1983.

G.S. TSEITIN : On the complexity of derivations in the propositional calculus. *In* H.A.O. SLESENKO, éditeur : *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

Alan TURING et Jean-Yves GIRARD : *La machine de Turing*. Édition du Seuil, collection source du savoir, 1991.

Tomás E. URIBE et Mark E. STICKEL : Ordered binary decision diagrams and the davis-putnam procedure. *In CCL '94 : Proceedings of the First International Conference on Constraints in Computational Logics*, pages 34–49, London, UK, 1994. Springer-Verlag. ISBN 3-540-58403-X.

Marc VAN DONGEN : Ac-3d an efficient arc-consistency algorithm with a low space-complexity. *In* Pascal VAN HENTENRYCK, éditeur : *Principles and Practice of Constraint Programming - CP 2002*, volume 2470 de *Lecture Notes in Computer Science*, pages 159–184. Springer Berlin / Heidelberg, 2006.

Pascal VANDER-SWALMEN : *Aspects parallèles des problèmes de satisfaisabilité*. Thèse doctorat, Université de Reims Champagne-Ardenne en collaboration avec l'Université de Picardie Jules Verne, UFR Sciences Exactes et Naturelles (URCA), UFR des Sciences (UPJV) CReSTIC (URCA), MIS (UPJV), décembre 2009.

Michel VASQUEZ et Audrey DUPONT : Filtrage par arc-consistance et recherche tabou pour l'allocation de fréquences avec polarisation. *In Journées nationales sur la résolution pratique de problèmes NP-complets*, JFPC02, pages 225–237, 2002.

Miroslav N. VELEV et Randal E. BRYANT : Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. *J. Symb. Comput.*, 35:73–106, February 2003.

Julien VION : Hybridation de prouveurs CSP et apprentissage. *In Troisièmes Journées Francophones de Programmation par Contraintes (JFPC07)*, JFPC07, juin 2007.

Toby WALSH : Search in a small world. *In Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, pages 1172–1177, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

David L. WALTZ : Generating semantic descriptions from drawings of scenes with shadows. Rapport technique, Cambridge, MA, USA, 1972.

Lawrence WOS, George A. ROBINSON et Daniel F. CARSON : Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12:536–541, October 1965.

Ke XU, Fr´ed´eric BOUSSEMART, Fred HEMERY et Christophe LECOUTRE : A simple model to generate hard satisfiable instances. *In Proceedings of the 19th international joint conference on Artificial intelligence*, pages 337–342, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

Ke XU et Wei LI : Exact phase transitions in random constraint satisfaction problems. *Journal Artificial Intelligence Research*, 12:93–103, March 2000.

Lin XU, Frank HUTTER, Holger H. HOOS et Kevin LEYTON-BROWN : Satzilla : portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008. ISSN 1076-9757.

Nail' K. ZAMOV et V. I. SHARONOV : On a class of strategies for the resolution method (in russian). *Studies in constructive mathematics and mathematical logic. Part III*, 16:54–64, 1969.

Hantao ZHANG : Sato : An efficient prepositional prover. *In* William MCCUNE, éditeur : *Automated Deduction–CADE-14*, volume 1249 de *Lecture Notes in Computer Science*, pages 272–275. Springer Berlin / Heidelberg, 1997.

Hantao ZHANG et Maria Paola BONACINA : Cumulating search in a distributed computing environment : A case study in parallel satisfiability. *In Proceedings of the First Intelligence Symposium on Parallel Symbolic Computation*, pages 422–431. Publishing Company, 1994.

Hantao ZHANG, Maria Paola BONACINA et Jieh HSIANG : Psato : a distributed propositional prover and its application to quasigroup problems. *J. Symb. Comput.*, 21:543–560, June 1996.

Hantao ZHANG et Mark E. STICKEL : An efficient algorithm for unit propagation. *In Proceedings of Mathematics and Artificial Intelligence Symposium*, Fort Lauderdale (FL USA), janvier 1996.

Jian ZHANG et Hantao ZHANG : Combining local search and backtracking techniques for constraint satisfaction. *In Proceedings of the Thirteenth American National Conference on Artificial Intelligence (AAAI'96)*, pages 369–374, juillet 1996.

Lintao ZHANG, Conor F. MADIGAN, Matthew H. MOSKEWICZ et Sharad MALIK : Efficient conflict driven learning in a boolean satisfiability solver. *In ICCAD '01 : Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 279–285, Piscataway, NJ, USA, novembre 2001. IEEE Press. ISBN 0-7803-7249-2.

Lintao ZHANG et Sharad MALIK : The quest for efficient boolean satisfiability solvers. *In CADE-18 : Proceedings of the 18th International Conference on Automated Deduction*, pages 295–313, London, UK, 2002. Springer-Verlag. ISBN 3-540-43931-5.

Yuanlin ZHANG et Roland H. C. YAP : Making ac-3 an optimal algorithm. *In Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1*, pages 316–321, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

Résumé

La thèse porte sur la résolution des problèmes de satisfiabilité propositionnelle (SAT) et des problèmes de satisfaction de contraintes (CSP). Ces deux modèles déclaratifs sont largement utilisés pour résoudre des problèmes combinatoires de première importance comme la vérification formelle de matériels et de logiciels, la bioinformatique, la cryptographie, la planification et l'ordonnancement de tâches. Plusieurs contributions sont apportées dans cette thèse. Elles vont de la proposition de schémas d'hybridation des méthodes complètes et incomplètes, répondant ainsi à un challenge ouvert depuis 1998, à la résolution parallèle sur architecture multi-cœurs, en passant par l'amélioration des stratégies de résolution. Cette dernière contribution a été primée à la dernière conférence internationale du domaine (prix du meilleur papier). Ce travail de thèse a donné lieu à plusieurs outils (open sources) de résolution des problèmes SAT et CSP, compétitifs au niveau international.

Mots-clés: SAT, CSP, hybridation, résolution parallèle

Abstract

This thesis deals with propositional satisfiability (SAT) and constraint satisfaction problems (CSP). These two declarative models are widely used for solving several combinatorial problems (e.g. formal verification of hardware and software, bioinformatics, cryptography, planning, scheduling, etc.). The first contribution of this thesis concerns the proposition of hybridization schemes of complete and incomplete methods, giving rise to an original answer to a well-known challenge open since 1998. Secondly, a new and efficient multi-core parallel approach is proposed. In the third contribution, a novel approach for improving clause learning management database is designed. This contribution allows spatial complexity reduction of the resolution-based component of SAT solvers while maintaining relevant constraints. This contribution was awarded at the last international SAT conference (best paper award). This work has led to several open sources solving tools for both propositional satisfiability and constraints satisfaction problems.

Keywords: SAT, CSP, hybridization, parallel solver