Ajustement dynamique de l'heuristique de polarité dans le cadre d'un solveur SAT parallèle

Long Guo **Jean-Marie Lagniez**

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens

{guo,lagniez}@cril.univ-artois.fr

Résumé

Dans cet article, nous proposons une nouvelle heuristique pour la polarité d'affectation d'une variable, dans le cadre d'un solveur SAT parallèle. La polarité selon laquelle le prochain point de choix sera 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. Dans ce cas, la polarité peut être utilisée pour diriger le solveur dans l'espace de recherche. Nous proposons un critère basé sur le *progress saving* afin d'évaluer si deux solveurs étudient le même espace de recherche. Une fois ce critère établi, nous l'utilisons pour attribuer une heuristique de polarité aux différents solveurs de manière dynamique. Des résultats prometteurs ont été obtenus et montrent l'intérêt de notre approche sur l'affectation de polarité dynamique dans le cadre de solveurs SAT parallèles de type portfolio.

Abstract

In this paper, a new heuristic for polarity selection is proposed. This heuristic is defined for parallel SAT solvers. The selected polarity is an important component of modern SAT solvers, more particulary in portfolio SAT solvers. Indeed, these solvers are often based on the cooperation/competition principle. In this case, the polarity can be used to guide the solver in the search space. A criterion based on the progress saving is proposed in order to evaluate if two solvers study the same search space. Once this criteria defined, a dynamical heuristic polarity is proposed for tuning the different solvers. Experimental results shows that our approach is efficient and increases the capacity of the parallel SAT solver.

1 Introduction

Le problème SAT est un problème central en intelligence artificielle, que ce soit d'un point de vue pratique (planification, vérification formelle, bio-informatique, etc) que d'un point de vue théorique (premier problème à avoir été montré NP-complet). D'un point de vue pratique les performances des solveurs n'ont cessé d'évoluer [13], permettant de résoudre des problème de plus en plus conséquents (des millions de variables et de contraintes). Cette évolution est due, conjointement, à l'avènement des solveurs SAT modernes [16] (*watched literal* , VSIDS, apprentissage) et à l'augmentation de la puissance de calcul (doublant tous les deux ans). Notre époque est témoin d'une « révolution » technologique, qui réside dans l'arrivée des architectures parallèles, permettant d'ouvrir de nouvelles perspectives.

Deux types d'approche pour résoudre SAT en parallèle sont principalement utilisés à l'heure actuelle. D'une part, les approches de type « diviser pour régner » consistent à partager l'arbre de recherche (« *guiding path* ») [14]. D'autre part, les approches de type portfolio [9] mettent les solveurs en concurrence, permettant ainsi de résoudre une formule à l'aide de différentes stratégies. C'est sur ce type d'approche que s'articulent les travaux proposés dans cet article.

Les solveurs SAT modernes sont très sensibles aux réglages de leurs paramètres initiaux [4]. Par exemple, sur certaines instances, changer les paramètres relatifs à la fréquence des redémarrages ou à l'heuristique de choix de polarité peut fortement affecter les performances d'un solveur. Dans ce contexte, une approche de type portfolio permet d'exécuter différentes versions d'un même solveur sur une même instance, profitant ainsi au maximum des ca-

pacités des solveurs SAT modernes.

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. 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 à ce problème, nous proposons dans cet article une mesure permettant d'estimer le comportement d'un solveur vis-à-vis d'un autre. Cette mesure est basée sur la notion de *progress saving* [17]. L'intuition est que l'interprétation représentant le *progress saving* peut être vue comme une image de la recherche en cours. De cette manière, deux solveurs peuvent être considérés comme proches s'ils explorent de la même manière l'espace de recherche. Une fois cette mesure définie, celle-ci est 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.

L'article est organisé de la façon suivante. Après quelques définitions préliminaires (section 2), nous présentons le schéma de base d'un solveur CDCL (*Conflict Driven Clause Learning*) ainsi que deux composantes importantes les caractérisant (heuristique de polarité et stratégie de redémarrage). Une fois ces élément définis, nous présentons le solveur MANYSAT1.1 [9] sur lequel nous avons appliqué notre méthode (section 4). Dans la section 5, nous présentons notre mesure permettant d'estimer si deux solveurs effectuent un travail redondant. Ensuite, une méthode utilisant cette mesure est proposée pour ajuster de manière dynamique l'heuristique de polarité associée à un solveur. Avant de conclure et de donner quelques perspectives, nous donnons des résultats expérimentaux dans la section 6.

2 Définitions et notations

Le problème SAT est le problème de décision qui consiste à vérifier si une formule booléenne Σ sous Forme Normale Conjonctive (CNF), possède ou non une solution. Une formule CNF Σ est un ensemble (interprété comme une conjonction) de *clauses*, où une clause $c_i = (\ell_1 \vee \cdots \vee \ell_{n_i})$ est un ensemble (interprété comme une disjonction) de *littéraux*. Un littéral ℓ est soit une variable positive x ou négative $\neg x$. Le littéral opposé de ℓ est noté $\overline{\ell}$. L'ensemble des variables appartenant à Σ sera noté $\mathcal{V}_\Sigma$. Une interprétation $\mathcal{I}$ d'une formule booléenne Σ associe une valeur $\mathcal{I}(x)$ aux variables $x \in \mathcal{V}_\Sigma$. L'interprétation $\mathcal{I}$ est dite *complète* si elle associe une valeur de vérité à chaque $x \in \mathcal{V}_\Sigma$, sinon elle est dite *partielle*. Une interprétation peut également être représentée par un ensemble de littéraux. Un *modèle* d'une formule de Σ est une interprétation $\mathcal{I}$ qui satisfait la formule, ce qui est noté $\mathcal{I} \models \Sigma$. Les notations suivantes seront utilisées par la suite :

- $\#occ(\ell)_\Sigma$ dénote le nombre d'occurrences du littéral ℓ dans la formule Σ (*i.e.* $|\{c \in \Sigma, \ell \in c\}|$). Lorsqu'il n'y a pas d'ambiguïté $\#occ(\ell)_\Sigma$ sera noté $\#occ(\ell)$;
- $\mathcal{C}_i$ dénote une unité de calcul (*i.e.* un solveur).

3 Notions préliminaires

Dans cette section, nous présentons le schéma général d'un solveur CDCL ainsi que deux composantes importantes de ce type de solveur (*i.e.* l'heuristique de choix de polarité et les différentes stratégies de redémarrage).

3.1 Solveur CDCL

Algorithme 1 : Solveur CDCL

Input : une formule CNF Σ
Output : SAT ou UNSAT

1 $\mathcal{I} = \emptyset$; /* interprétation */
2 $dl = 0$; /* niveau de décision */
3 **while** *(true)* **do**
4 γ = propagationUnitaire(Σ,$\mathcal{I}$);
5 **if** *(γ !=null)* **then**
6 β = analyseConflit(Σ,$\mathcal{I}$,γ);
7 bl = calculNiveauBackjumping(β,$\mathcal{I}$);
8 **if** *($bl < 0$)* **then return** UNSAT;
9 $\Sigma = \Sigma \cup \{\beta\}$;
10 **if** *(restart())* **then** $bl = 0$;
11 backjump(Σ,$\mathcal{I}$,bl);
12 $dl = bl$;
13 **else**
14 **if** *($\mathcal{I} \models \Sigma$)* **then**
15 **return** SAT;
16 **end**
17 ℓ = choisirVariableDecision(Σ);
18 $dl = dl + 1$;
19 $\mathcal{I} = \mathcal{I} \cup \{selectPhase(\ell)\}$;
20 **end**
21 **end**

L'algorithme 1 donne le schéma général d'un solveur de type CDCL. Typiquement, un tel solveur peut être assimilé à une séquence de décision et de propagation des littéraux unitaires. Chaque littéral choisi comme littéral de décision (lignes 18-20) est affecté suivant une certaine polarité à un niveau de décision (dl). Si tous les littéraux sont affectés, alors $\mathcal{I}$ est un modèle de Σ (lignes 16-17). À chaque fois qu'un conflit est atteint par propagation (γ est la clause conflit), un *nogood* β est calculé (ligne 8) en

utilisant une méthode d'analyse de conflits donnée, le plus souvent selon le schéma « First UIP » (*« Unique Implication Point »* [18]), et un niveau de *backjump bl* est calculé. À ce moment, il est possible de prouver l'inconsistance de la formule (ligne 10). Si ce n'est pas le cas, on fait un saut arrière et le niveau de décision devient égal au niveau de *backjump* (ligne 13-14). Finalement, certains solveurs CDCL forcent le redémarrage (diverses stratégies sont possibles, voir [11] par exemple) et dans ce cas, on remonte en haut de l'arbre (ligne 12).

3.2 Heuristique de sélection de polarité

L'heuristique de choix de variable est l'une des composantes les plus importantes des solveurs SAT modernes. La plus utilisée à l'heure actuelle est VSIDS (*Variable State Independent Decaying Sum*) [16]. Chaque fois qu'une nouvelle variable x est sélectionnée comme point de choix, il est nécessaire de lui associer une polarité (x ou $\neg x$). Dans le solveur MINISAT [7] une heuristique de phase très simple a été proposée. Celle-ci consiste à toujours affecter la variable à la valeur fausse (*false* : $\neg x$). Jeroslaow et Wang [12], quant à eux, 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 (x si $w(x) > w(\neg x)$, $\neg x$ sinon). Une manière simple d'utiliser cette heuristique est de considérer le nombre d'occurrences d'un littéral comme mesure (*occurence* : x si $\#occ(x) > \#occ(\neg x)$, $\neg x$ sinon). Dans [17], Pipatsrisawat et Darwish 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, noté $\mathcal{P}$. Ainsi, lorsqu'une nouvelle décision sera prise, la variable se verra attribuée la même polarité que précédemment (*progress saving* : 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.

3.3 Différentes stratégies de redémarrage

La politique de redémarrage est une composante importante des solveurs SAT modernes [8]. Sommairement, le principe consiste à remonter l'arbre de recherche jusque la racine, tout en conservant les informations obtenues jusqu'ici (pondération des variables, clauses apprises, ...), lorsqu'un certain nombre de conflits a été atteint. Différentes politiques de redémarrage ont été proposées. Dans le cadre de cet article, nous ne présentons que les politiques de redémarrage utiles à la compréhension (pour plus d'information sur le sujet le lecteur pourra consulter [11]). Le nombre de conflits pouvant être rencontrés au $n^{\text{ème}}$ redémarrage (avant d'effectuer un retour-arrière à la racine) peut être calculé à l'aide des fonctions suivantes :

- $geometrical(x, y)$: stratégie dans lequelle on a un redémarrage tous les x conflits. La valeur x est multipliée par y à chaque redémarrage ;
- $luby(x)$: le $i^{\text{ème}}$ terme de la suite de Luby[15] se calcule comme suit : $t_i = 2^{k-1}$ si $i = 2^{k-1}$, sinon $t_i = t_{i-2^{k-1}+1}$. Le nombre de conflits à atteindre avant d'effectuer le $n^{\text{ème}}$ redémarrage est égal à $t_n \times x$;
- $dynamic^{+}(x_1, x_2)$: cette fonction est calculée comme suit : on a $x_i = y_i \times |cos(1 + r_i)|$, 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 ;
- $dynamic^{-}(x_1, x_2)$: cette fonction est une variante de $dynamic^{+}$ où $x_i = y_i \times |cos(1 - r_i)|$.

L'heuristique $dynamic$ a été proposée par Jabbour *et al.* et a été implémentée dans le solveur LYSAT [10].

4 Solveur parallèle

Il existe principalement deux types d'approches pour résoudre SAT en parallèle. D'une part, les approches de type « diviser pour régner » qui consistent à partager l'arbre de recherche (« *guiding path* ») [14]. D'autre part, les approches de type portfolio [9] où les solveurs sont mis en concurrence permettant ainsi de résoudre une formule à l'aide de différentes stratégies. Dans cette article, seule l'approche portfolio est présentée. Nous détaillons plus particulièrement MANYSAT1.1, qui est le solveur utilisé pour la partie expérimentale de nos travaux. La figure 1 représente de manière schématique le fonctionnement d'un solveur parallèle.

Comme présenté plus haut, MANYSAT1.1 est un solveur parallèle de type portfolio. Dans MANYSAT1.1, chaque coeur représente un solveur séquentiel contenant toutes les stratégies classiques (*i.e.* propagation unitaire, *watched litterals*, VSIDS, analyse de conflits, etc). De plus, lorsqu'un solveur apprend une clause (*i.e.* un conflit est atteint), celle-ci est transférée aux autres solveurs (si sa taille est inférieure à 8). Ce type d'approche est basé sur le principe de compétition/coopération. Compétition, dans le sens où les différents solveurs ont des stratégies différentes. Et coopération, dans le sens où les solveurs se partagent des informations par le biais des clauses apprises. Le tableau 1 détaille les différentes stratégies utilisées dans MANYSAT1.1. En ce qui concerne la stratégie d'apprentissage utilisée par le coeur 3, celle-ci a été proposée dans [2] et est basée sur le principe d'extension du graphe d'implication classique à l'aide d'arcs inverses. Ces arcs inverses, sont obtenus en prenant en compte les clauses satisfaites dans l'analyse du graphe d'implication.

Stratégie	Coeur 1	Coeur 2	Coeur 3	Coeur 4
Redémarrage	$dynamic^+$	$dynamic^-$	$geometrical(100, 1.5)$	$dynamic^+$
Heuristique	VSIDS (2% rand.)	VSIDS (2% rand.)	VSIDS (2% rand.)	VSIDS (3% rand.)
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 1 – Stratégie des différents solveurs séquentiels utilisés dans le solveur MANYSAT1.1.

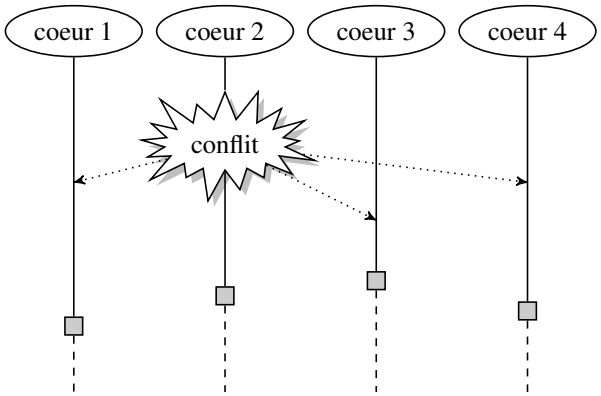

FIGURE 1 – Présentation schématique d'un solveur parallèle de type portfolio. Les lignes continues représentent l'exécution d'un solveur. Les carrés symbolisent les redémarrages tandis que les flèches en pointillés représentent le partage d'information entre les différents solveurs (ici les clauses apprises).

5 Utilisation du *progress saving* pour estimer la distance entre solveurs

Comme précisé précédemment, les solveurs parallèles de types portfolio sont souvent basés sur le principe de compétition/coopération. Ainsi dans le solveur MANYSAT1.1, 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, ...) pour chacun des solveurs. Contrairement à la partie coopérative du solveur, la manière dont les solveurs entrent en compétition est choisi de manière statique au début de la recherche. Ce type d'approche ne permet pas d'identifier ni de traiter le cas où deux solveurs effectuent le même travail. En effet, bien que deux solveurs aient 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 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 proposé. Pour cela, une mesure permettant d'estimer la distance entre deux solveurs est introduite. Afin d'étudier le comportement des solveurs vis-à-vis de notre mesure, nous avons effectué un ensemble d'expérimentations.

5.1 Distance entre deux solveurs

Pour savoir si deux solveurs sont en train d'effectuer le même travail, nous avons choisi d'utiliser le *progress saving* $\mathcal{P}$. En effet, il est possible de considérer l'interprétation complète $\mathcal{P}$ comme une image de la recherche en cours. De cette manière, on peut supposer que deux solveurs sont voisins dans l'espace de recherche lorsqu'ils ont des *progress saving* proches (proche au sens de distance de Hamming). De cette analyse, il est possible de définir une mesure permettant d'estimer la distance entre deux solveurs. Cette mesure sera notée δ.

Définition 1 *Soient Σ une formule* CNF, *$(\mathcal{C}_i, \mathcal{P}_i)$ et $(\mathcal{C}_j, \mathcal{P}_j)$ deux couples de solveurs avec leurs progress savings respectifs, on note :* $\delta(\mathcal{C}_i, \mathcal{C}_j) = 1 - \frac{|\mathcal{P}_i \cap \mathcal{P}_j|}{|\mathcal{V}_\Sigma|}$.

Remarquons que deux solveurs sont proches, d'après notre mesure, si leurs interprétations représentant leurs *progress savings* respectifs ont une petite distance de hamming.

5.2 Évolution de la distance entre différents solveurs

Afin d'étudier l'évolution de la distance entre différents solveurs entre eux, nous avons exécuté le solveur MANYSAT1.1 en récupérant la distance entre les différents solveurs tous les 5000 conflits. Les courbes de la figure 2 retranscrivent, de manière représentative, le comportement des différents solveurs les uns envers les autres. On peut constater que les solveurs se divisent en deux catégories. La première catégorie regroupe les courbes où le coeur 4 apparaît et la seconde catégorie les autres. Concernant les courbes de la première catégorie, on observe que la distance entre le coeur 4 et n'importe quel autre coeur est toujours supérieure au cas où le coeur 4 n'est pas présent dans

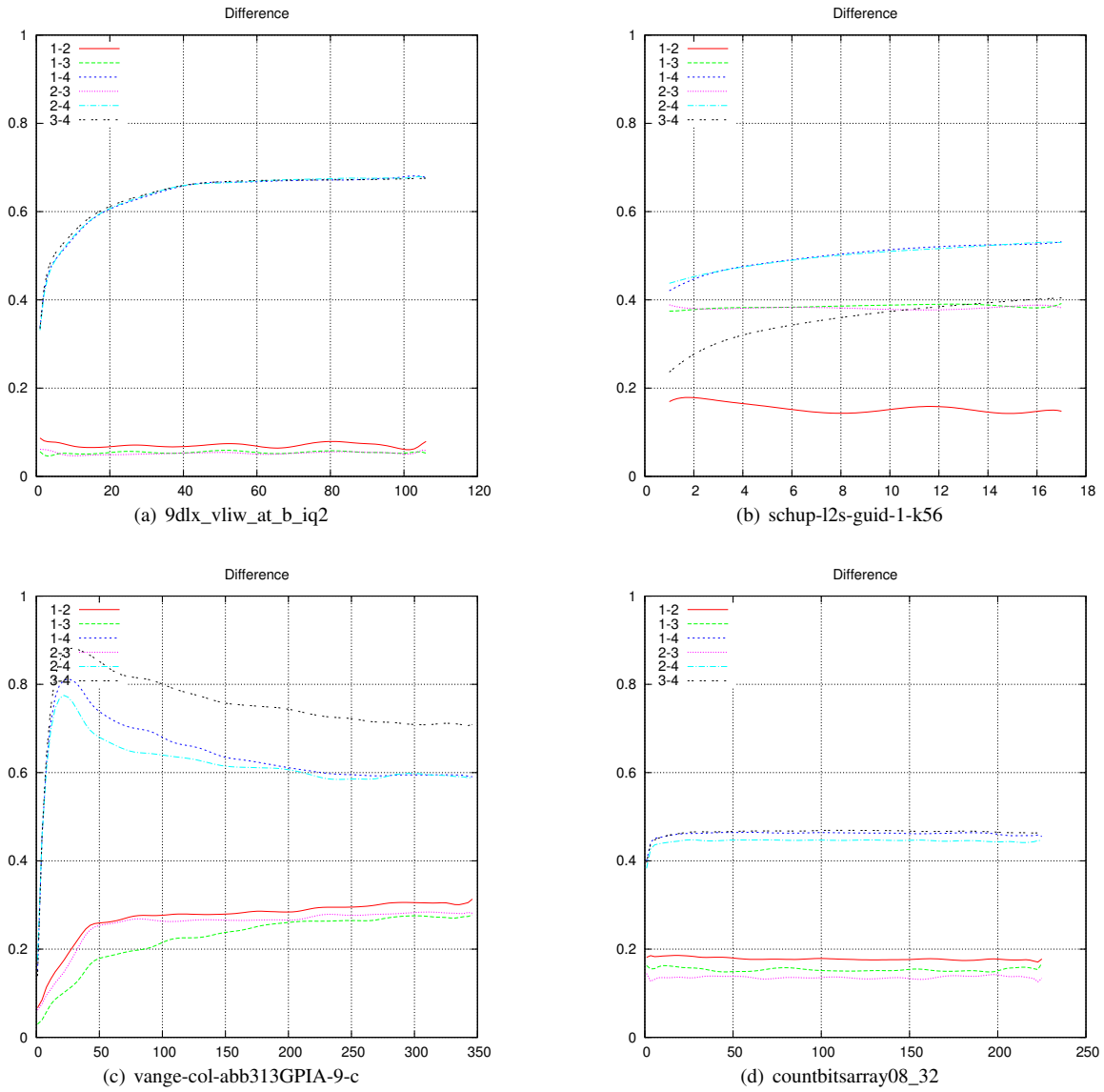

FIGURE 2 – Comparaison deux à deux de la distance entre différents solveurs exécutés en parallèle. Sur l'axe des abscisses est représenté le nombre de conflits atteint (divisé par 5000) tandis que l'axe des ordonnées représente le résultat de notre mesure. Pour une meilleure lisibilité toutes les courbes ont été lissées.

le calcul de la distance. Ceci peut être expliqué par le choix de la stratégie d'affectation de polarité du coeur 4. En effet, son heuristique de choix de polarité est basée sur le nombre d'occurrences d'un littéral dans la formule. Étant donné que les solveurs s'échangent de nombreuses clauses, il y a une forte variation du nombre d'occurrences d'un littéral, ceci a pour conséquence de grandement diversifier l'espace de recherche exploré par le coeur 4. Pour se qui est de la seconde catégorie, on peut voir que les coeurs 1, 2 et 3 sont beaucoup plus proches les uns des autres. Ceci est facilement explicable pour les coeur 1 et 2, étant donné qu'ils utilisent tous deux la même heuristique de choix de polarité. En ce qui concerne le coeur 3, nous pensons que cela est du au choix de la polarité initiale pour le *progress saving* (la phase étant affectée à « faux »).

5.3 Ajustement dynamique de la polarité

Après 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 coeurs sont détectés comme « trop proches », nous choisissons d'inverser la polarité d'un des

deux coeurs. 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, cette ajustement ne sera pas fait de manière systématique. La figure 3 décrit le transfert de messages et l'ajustement de la polarité entre deux solveurs. À chaque point de contrôle le coeur b va demander au coeur a s'ils sont proches. Le coeur a lui répond par oui ou par non en fonction du résultat obtenu en mesurant sa distance avec le coeur b à l'aide de notre mesure. Dans le cas où la réponse est négative, le coeur b utilise son heuristique de choix de polarité initial. Dans le cas d'une réponse positive, lorsque le coeur b sera de nouveau amené à choisir la polarité d'un point de choix, il inversera simplement son heuristique de polarité. Par exemple, supposons que le coeur b aie comme heuristique de polarité l'heuristique *false*. Dans ce cas, tant qu'un nouveau point de contrôle n'est pas atteint, le solveur affectera les prochains points de choix à vrai. Une fois le point de contrôle atteint, 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.

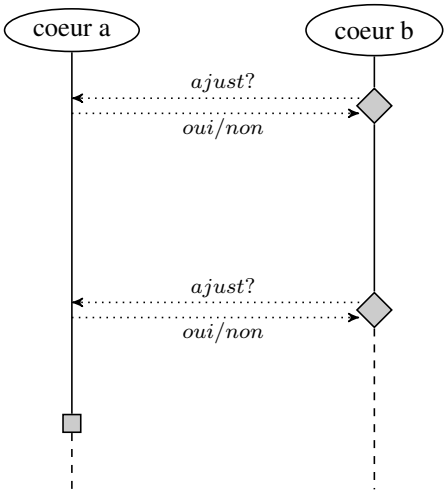

FIGURE 3 – Ajustement dynamique de la polarité d'un solveur vis-à-vis d'un autre.

6 Expérimentations

Afin d'étudier l'impact de notre méthode, les expérimentations ont été divisées en deux parties. Dans la première, 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, l'ensemble des unités de calcul utilisent des architectures identiques (*i.e.* même politique de redémarrage, même stratégie d'analyse de conflit et même fréquence du choix aléatoire de la prochaine variable à affecter). Dans la seconde partie, nous avons intégré directement notre méthode au solveur MANYSAT1.1.

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 entre eux, notre méthode ne sera pas appliquée sur le coeur 4 (coeur utilisant l'heuristique de polarité *occurence*). Ensuite, afin d'éviter des problèmes de cycle dans le protocole d'ajustement de polarité notre approche ne sera pas non plus appliquée au coeur 1. La figure 4 schématise la manière selon laquelle l'ajustement des coeurs 2 et 3 sera effectué. Nous avons choisi d'ajuster le coeur 2 dans le cas où il est proche ($\delta(\mathcal{C}_1, \mathcal{C}_2) \leq 0.1$) du coeur 1. L'heuristique de polarité du coeur 3 sera ajustée dans le cas où $\delta(\mathcal{C}_1, \mathcal{C}_3) \leq 0.1$ et que $\delta(\mathcal{C}_1, \mathcal{C}_2) > 0.1$. Les contrôles seront effectués tous les 5000 conflits.

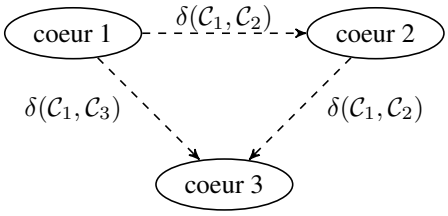

FIGURE 4 – Schéma d'ajustement.

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 dernière compétition SAT [1]. Toutes les instances sont pré-traitées à l'aide de SATElite [6]. Étant donnée le non déterminisme, lié à la résolution en parallèle du problème, chaque solveur sera lancé trois fois. Les résultats reportés sont selectionnés parmi le meilleur des trois lancements.

6.1 Architecture identique

Comme précisé précédemment, afin d'isoler l'heuristique de polarité des autres composantes des solveurs SAT modernes, nous avons choisi d'utiliser une architecture identique pour l'ensemble des coeurs. En ce qui concerne l'heuristique de polarité nous avons conservé celle implémentée dans la version initiale de MANYSAT1.1 (voir figure 1). Sur les figures 5 et 6, nous pouvons constater que, sur les résultats obtenus sur l'ensemble des instances industrielles de la dernière compétition SAT, de notre technique (74 SAT et 120 UNSAT) permet d'améliorer sensible-

ment les performances du solveurs (72 SAT et 111 UNSAT). Plus particulièrement, le nuage de point 6 nous montre que l'ajout de notre méthode permet toujours de résoudre de manière plus efficace les instances insatisfiables. Concernant les instances satisfiables nous pouvons noter, même si les points sont cette fois-ci plus éparpillés, que notre approche est la meilleure.

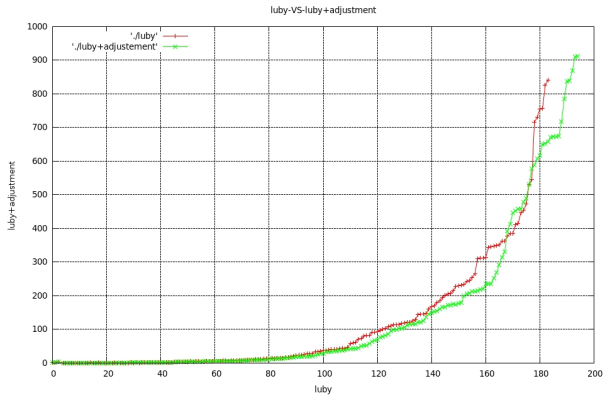

FIGURE 5 – Nombre d'instances résolues en fonction du temps pour le solveur parallèle MANYSAT1.1 intégrant ou pas notre technique d'ajustement de polarité, dans le cas d'une architecture identique pour tous les coeurs.

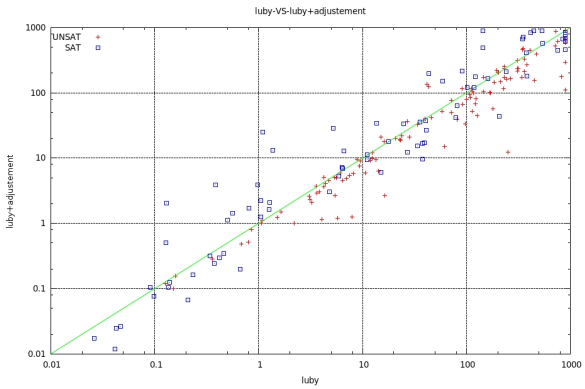

FIGURE 6 – Corrélation entre le temps mis par chacune des deux méthodes (MANYSAT1.1 architecture identique et MANYSAT1.1 architecture + ajustement) pour résoudre une instance donnée.

6.2 Ajustement de la polarité dans MANYSAT1.1

Ici nous évaluons le gain apporté par notre technique d'ajustement de polarité dans le cadre du solveur MANYSAT1.1, lequel résout 73 instances satisfiables et 120 instances insatisfiables. Les résultats obtenus et reportés sur les figures 7 et 8 montrent que l'ajout de notre méthode au sein du solveur SAT parallèle MANYSAT1.1 permet d'améliorer sa compétitivité (81 SAT et 123 UNSAT). Comme dans le cas où les architectures sont identiques, nous pouvons voir que l'ajout de notre méthode permet au solveur de résoudre plus efficacement les instances insatisfiables. En ce qui concerne les instances satisfiable les résultats montrent que notre approche permet de résoudre sensiblement plus d'instances.

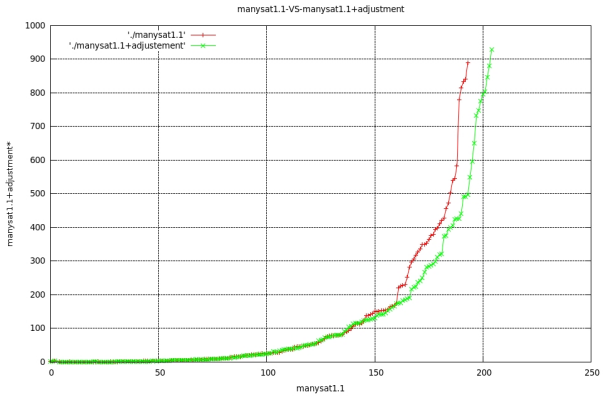

FIGURE 7 – Nombre d'instances résolues en fonction du temps pour le solveur parallèle MANYSAT1.1 intégrant ou pas notre technique d'ajustement de polarité.

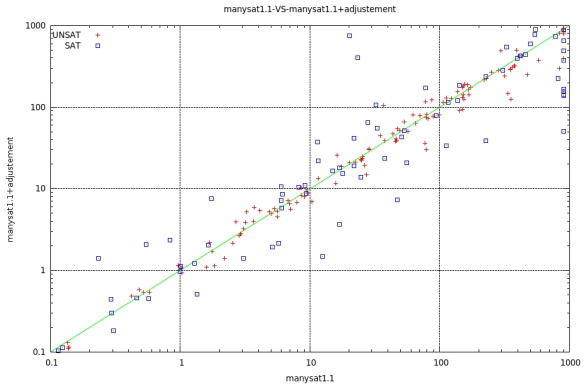

FIGURE 8 – Corrélation entre le temps mis par chacune des deux méthodes (MANYSAT1.1 et MANYSAT1.1 + ajustement) pour résoudre une instance donnée.

7 Conclusion et perspectives

Des travaux sur l'utilisation du *progress saving* pour régler certaines composantes des solveurs SAT modernes ont déjà été menés par le passé. Ainsi, dans [5], une politique de redémarrage basée sur le *progress saving* est proposée. Plus récemment, Audemard *et al.* [3] ont proposé d'utiliser cette notion pour contrôler la base de clauses apprises. À notre tour, en utilisant une approche d'ajustement très simple basé sur le *progress saving*, nous avons pu améliorer les performances d'un solveur SAT de type portfolio.

Cette approche consiste à régler de manière dynamique l'heuristique de choix de polarité. Notre méthode permet d'estimer la distance séparant deux solveurs à l'aide du *progress saving*. L'avantage de cette mesure est de fournir des informations sur les différents coeurs, permettant ainsi d'agir sur l'heuristique de choix de polarité pendant la recherche. Les résultats expérimentaux, menés sur l'ensemble des instances industrielles de la dernière compétition SAT, ont montré que notre méthode améliore sensiblement les performances du solveur MANYSAT1.1 auquel a été greffée notre méthode. Une étude du comportement de notre approche sur les autres familles d'instances est envisagée. 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.

Au lieu de se réduire à inverser l'heuristique de choix de polarité comme nous l'avons fait ici, une autre approche pourrait consister à en changer complètement pendant la phase de recherche. De cette manière un portfolio d'heuristique de polarité serait obtenu. Une tout autre perspective consisterait à utiliser les informations fournies par le *progress saving* pour contrôler les flux de clauses apprises transitant entre les différents coeurs d'un solveur parallèle.

Remerciements

Nous tenons à remercier Elodie Vandenhaute et Gilles Audemard pour les commentaires qu'ils nous ont apportés sur la rédaction de cet article.

Références

[1] « 12th international symposium on theory and applications of satisfiability testing ». May 2009.

[2] Gilles Audemard, Lucas Bordeaux, Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. A generalized framework for conflict analysis. In *SAT*, pages 21–27, 2008.

[3] Gilles Audemard, Jean Marie Lagniez, Bertrand Mazure, and Lakdhar Saïs. RCL : reduce learnt clauses. SATRACE, solver description, 2010.

[4] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[5] Armin Biere. Adaptive restart strategies for conflict driven sat solvers. In *SAT*, pages 28–33, 2008.

[6] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT*, pages 61–75, 2005.

[7] Niklas Eén and Niklas Sörensson. An extensible satsolver. In *SAT*, pages 502–518, 2003.

[8] Carla P. Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff. Randomization in backtrack search : Exploiting heavy-tailed profiles for solving hard scheduling problems. In *AIPS*, pages 208–213, 1998.

[9] Youssef Hamadi, Saïd Jabbour, and Lakhdar Sais. Manysat : a parallel sat solver. *JSAT*, 6(4) :245–262, 2009.

[10] Youssef Hamadi, Saïd Jabbour, and Lakhdar Saïs. Lysat : Solver description. SATRACE, solver description, 2010.

[11] J. Huang. The effect of restarts on the efficiency of clause learning. In *proceedings of IJCAI*, pages 2318–2323, 2007.

[12] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Ann. Math. Artif. Intell.*, 1 :167–187, 1990.

[13] Daniel Le Berre. Sat4j, un moteur libre de raisonnement en logique propositionnelle, dec 2010.

[14] Matthew D. T. Lewis, Tobias Schubert, and Bernd Becker. Multithreaded sat solving. In *ASP-DAC*, pages 926–931, 2007.

[15] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal speedup of las vegas algorithms. In *ISTCS*, pages 128–133, 1993.

[16] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *DAC*, pages 530–535, 2001.

[17] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *SAT*, pages 294–299, 2007.

[18] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.