

# La méthode d'avalanche AVAL : Une méthode énumérative pour SAT

G. Audemard, B. Benhamou et P. Siegel

Laboratoire d'Informatique de Marseille  
Centre de Mathématiques et d'Informatique  
39, Rue Joliot Curie - 13453 Marseille cedex 13  
Tel: 04 91 11 36 22 - Fax : 04 91 11 36 02  
email: {audemard, benhamou, siegel}@gyptis.univ-mrs.fr

## Résumé

Le travail que nous présentons dans cet article porte sur l'élaboration d'un algorithme basé sur la procédure de Davis & Putnam. Celui-ci consiste à produire et utiliser des monolitéraux (clauses à un seul littéral) non explicites que la méthode de Davis & Putnam n'exploite pas. En effet, en l'absence d'un monolittéral, notre méthode essaie de déduire intelligemment un littéral apparaissant dans une clause binaire pour exploiter au maximum la propriété du monolittéral qui fait la force de la méthode de Davis & Putnam. Une technique de recherche de monolitéraux est étudiée et utilisée pendant le processus d'énumération pour réduire la taille de l'arbre de recherche. On parlera d'algorithme d'avalanche "AVAL" car à partir d'une certaine profondeur de l'arbre de recherche on retrouve une cascade de monolitéraux réalisant un phénomène d'avalanche qui permet de terminer la recherche en temps polynomial. Cette méthode distingue deux parties dans l'arbre de recherche des problèmes aléatoires : La partie supérieure, difficile, où sont faits la plupart des points de choix, et la partie inférieure, facile, réduite aux propagations de monolitéraux.

**Mots-Clés** : Satisfaisabilité, déduction, énumération....

## 1 Introduction

Des progrès ont été réalisés sur la résolution du problème SAT. Notamment l'application des méthodes dites de recherche locale [1] aux instances difficiles du problème SAT satisfaisable a donné des résultats très satisfaisants. La difficulté demeure toujours pour les instances non satisfaisables situées dans la région difficile. C'est le point faible des méthodes de recherche locale qui ne s'appliquent pas dans ce cas. Pour résoudre ces problèmes, on utilise des méthodes de recherche systématique (complètes). La méthode de Davis & Putnam [6] (abréviation DP) est l'une des plus connues et utilisées pour résoudre SAT. Son efficacité est due en grande partie à la propriété de propagation des monolitéraux présents dans la base de clauses. Cette procédure, ainsi que ses améliorations connues ([11], [8], [13], etc) ont trouvé leurs limites face à la résolution des instances SAT aléatoires dans la région des problèmes difficiles.

Le travail que nous présentons consiste à l'étude et l'élaboration d'une méthode énumérative qui utilise au maximum la propagation des monolitéraux. Pour ce faire, notre algorithme déduit intelligemment des monolitéraux qui ne sont pas présents explicitement dans la base de clauses et que la méthode de Davis & Putnam n'exploite pas. Leur utilisation permettra de réduire l'espace de recherche.

Nous étudions un algorithme efficace de production de monolitéraux basé sur des propriétés théoriques que nous démontrons. Dans la pratique cet algorithme favorise la déduction des littéraux apparaissant dans les clauses binaires. En effet, ces littéraux ont plus de chance d'être conséquence logique de l'ensemble de clauses courant. La procédure de détection de monolitéraux est appelée chaque fois que la propriété classique des monolitéraux utilisée dans la méthode Davis & Putnam ne s'applique pas. La production d'un littéral permet d'éviter la création d'un point de choix et donc réduit la taille de l'arbre de recherche pendant l'énumération.

Le but de ce travail est double : d'une part, améliorer l'efficacité des méthodes d'énumération par l'utilisation efficace et optimale de la propagation des monolitéraux pour élaguer l'arbre de recherche. De l'autre, présenter un algorithme robuste qui permet d'expliquer les limites des méthodes complètes sur la résolution des problèmes SAT difficiles.

On parlera d'algorithme d'avalanche, car à partir d'une certaine profondeur de l'arbre de recherche (dans la pratique égal à  $\frac{n}{21}$  où  $n$  désigne le nombre de variables) on retrouve une cascade de monolitéraux réalisant un phénomène d'avalanche qui permet de finir la recherche de façon polynomiale. On arrive ainsi, dans la pratique, à une borne inférieure des noeuds devant être obligatoirement parcourus par une procédure d'énumération de type DP.

Cet article est organisé comme suit. Dans la section 2 on étudie l'algorithme de production des monolitéraux. La section 3 décrit la méthode d'avalanche AVAL, les heuristiques et les prétraitements utilisés (résolvantes sur les clauses). La section 4 présente les résultats des expérimentations faites sur une grande variété de problèmes : les problèmes aléatoires dans la région difficile et les problèmes des challenges Dimacs et Beijing. Une comparaison avec d'autres algorithmes parmi les plus performants (POSIT [8] et SATZ [10]) est faite. La section 5 conclue le travail.

## 2 Production des monolitéraux

Commençons tout d'abord par quelques définitions utiles. Soit un ensemble de variables booléennes  $V = \{x_1 \dots x_n\}$ , un littéral  $l$  est une variable  $x_i$  ou sa négation  $\bar{x}_i$ . Une clause est une disjonction de littéraux :  $c_i = l_1 \vee l_2 \dots \vee l_{n_i}$ . La forme conjonctive normale d'une formule propositionnelle  $C$  est une conjonction de clauses  $C = c_1 \wedge c_2 \dots \wedge c_n$ . On peut aussi considérer  $C$  comme l'ensemble de clauses  $C = \{c_1, \dots, c_n\}$ . Le problème SAT est le problème de décision suivant : existe-t-il une interprétation des variables de telle sorte que la formule  $C$  soit satisfaite, autrement dit, de telle sorte que toutes les clauses de l'ensemble  $C$  soient satisfaites. Si  $l$  est impliqué par l'ensemble de clauses  $C$ , on écrit alors  $C \models l$ . De même, lorsque un système de clauses  $C$  est inconsistant, on le note par  $C \models \square$  où  $\square$  désigne la clause vide.

Le problème  $k$ -SAT est le problème SAT où toutes les clauses ont exactement  $k$  littéraux. 3-SAT est la forme la plus simple de  $k$ -SAT qui reste NP-Complet.

Grâce à l'ajout de la propriété des monolitéraux et des littéraux purs (cf proposition 1), la méthode de Davis et Putnam a été une réelle amélioration de la méthode de Quine [12]. Dans la même idée, notre travail consiste à faire apparaître, à faible coût, des clauses unitaires que DP ne considère pas, afin de réduire la taille de l'arbre de recherche et le temps d'exécution.

La propriété des monolitéraux et des littéraux purs utilisée dans DP est exprimée par la proposition suivante.

**Proposition 1** *Soit  $S$  un problème SAT et  $x$  un monolittéral ou un littéral pur, alors  $S$  est consistant si et seulement si  $S \wedge \{x\}$  est consistant.*

A un noeud donné de l'arbre de recherche et en l'absence de monolittéral, notre algorithme essaie de produire intelligemment un monolittéral. Autrement dit, Il n'est pas souhaitable de parcourir tout l'arbre de recherche pour déduire ce littéral. Pour des raisons d'efficacité, on se limite à la production des littéraux apparaissant dans les clauses binaires. Logiquement, ce sont ces littéraux-là qui ont le plus de chance d'être impliqués par les propagations des clauses unitaires.

Soit  $I$  l'instantiation courante,  $C_I$  est l'ensemble des clauses  $C$  simplifié par l'instantiation  $I$ . Si  $l$  est un littéral apparaissant dans une clause binaire de  $C_I$ , on essaie alors de prouver  $C_I \models l$ . Ceci d'après le théorème de déduction revient à prouver l'inconsistance de  $C_I \wedge \{\neg l\}$ . Deux cas se présentent :

- Soit  $C_I \wedge \{\neg l\} \models \square$ , dans ce cas  $l$  est impliqué par  $C_I$  et est traité comme un monolittéral :  $I = I \cup \{l\}$
- Soit  $C_I \wedge \{\neg l\} \not\models \square$ , dans ce cas  $l$  n'est pas impliqué par  $C_I$ , mais cet échec a mis en valeur plusieurs littéraux qui ne pourront être impliqués par  $C_I$ . Plus précisément, ces littéraux sont tous les opposés des littéraux propagés lors de la tentative de déduction du littéral  $l$  qui a échoué. il sera donc inutile de les considérer dans les prochains choix des littéraux pour la production.

Procédure RLI( $C$  : ensemble de clauses ;  $I$  : interprétation)

**Sortie** : Un littéral  $l$  si  $C \models l$

0 Si  $\nexists l \in V_{B_I}$  tel que  $C \models l$

**Début**

**Pour Chaque**  $l \in V_{B_I}$  **Faire** Marque[ $l$ ]=Faux

**Pour Chaque**  $l \in V_{B_I}$  tel que Marque[ $\neg l$ ]=Faux **Faire**

**Début**

$I' = I \cup \{\neg l\}$

**Tant Que**  $C_{I'} \neq \emptyset$  et  $\exists x \in V_{C_{I'}}$  monolitéral et  $\square \notin C_{I'}$

**Faire**

**Début**

$I' = I' \cup \{x\}$

Marque[ $x$ ]=Vrai

**Fin**

**Si**  $\square \in C_{I'}$  **Alors Retourner**  $l$

**Fin**

**Retourner** 0

**Fin**

FIG. 1 – Algorithme de recherche des littéraux impliqués (RLI)

L'efficacité et la complexité de notre algorithme de déduction de monolitéraux dépend de l'élimination de ces variables inutiles. Formellement :

**Proposition 2** Soit  $V_{B_I}$  l'ensemble des littéraux apparaissant des clauses binaires de  $C_I$  ( $B_I$  est l'ensemble des clauses binaires de  $C_I$ ). Soit  $l \in V_{B_I}$ , si  $C_I \cup \{\neg l\} \models a_{i \in \{1..n\}}$  et  $C_I \cup \{\neg l\} \not\models \square$  alors  $\forall i \in \{1, 2, \dots, n\} C_I \cup \{a_i\} \not\models \square$ .

**Preuve**

Soit  $l \in V_{B_I}$ , supposons que  $C_I \cup \{\neg l\} \models a_{i \in \{1..n\}}$  et  $C_I \cup \{\neg l\} \not\models \square$

Supposons de plus qu'il existe  $a_j$  ( $j \in \{1..n\}$ ) tel que  $C_I \cup \{a_j\} \models \square$ .

On a donc  $C_I \cup \{\neg l\} \cup \{a_j\} \models \square$ .

Or  $C_I \cup \{\neg l\} \cup \{\neg a_j\} \models \square$ .

Et par conséquent  $C_I \cup \{\neg l\} \models \square$ . Ce qui est contradictoire avec l'hypothèse.

Donc  $\forall i \in \{1, 2, \dots, n\} C_I \cup \{a_i\} \not\models \square$ .

Les littéraux  $\{\neg a_1, \dots, \neg a_n\}$  ne peuvent pas être conséquence logique de  $C_I$ . Il est inutile d'essayer de les produire. La propriété exprimée par la proposition 2 se traduit par l'algorithme de recherche des littéraux impliqués (noté RLI) décrit dans la figure 1.

**Remarque 1** Dans l'algorithme décrit par la figure 1, Marque[ $x$ ]=Faux exprime le fait que  $\neg x$  est candidat à la déduction.

L'algorithme RLI permet de déduire un littéral parmi ceux apparaissant dans les clauses binaires. Sa terminaison, sa correction, sa complétude et sa complexité sont données par les propositions suivantes.

**Proposition 3 (Correction et complétude)** Soit  $C$  l'ensemble de clauses courant, et  $l$  un littéral apparaissant dans une clause binaire. Si  $l$  est produit par RLI, alors  $C \models C \wedge \{l\}$ .

**Preuve**

Soit  $l$  un littéral renvoyé par la procédure RLI. On a alors  $C \wedge \{\neg l\} \models \square$ .

Or  $C \models (C \wedge l) \vee (C \wedge \{\neg l\})$ . Par conséquent  $C \models C \wedge \{l\}$ .

Procédure **AVAL**( $C$  : ensemble de clauses ;  $I$  : interprétation)  
**Sortie** : Vrai : Si  $C$  est satisfaisable par  $I$   
Faux Sinon  
**Début**  
**Si**  $C = \emptyset$  **Alors Retourner** Vrai  
**Si**  $C$  contient une clause vide **Alors Retourner** Faux  
**Si**  $C$  contient un monolitéral ou un littéral pur  $l$  **Alors Retourner**  $\text{AVAL}(C_{\{l\}}, I \cup \{l\})$   
 $q = \text{RLI}(C, I)$   
**Si**  $q \neq 0$  **Alors Retourner**  $\text{AVAL}(C_{\{q\}}, I \cup \{q\})$   
Choisir un littéral  $p \in C$  (moyennant une heuristique)  
**Si**  $\text{AVAL}(C_{\{p\}}, I \cup \{p\})$   
**Alors Retourner** Vrai  
**Sinon Retourner**  $\text{AVAL}(C_{\{\neg p\}}, I \cup \{\neg p\})$   
**Fin**

FIG. 2 – Méthode AVAL

**Proposition 4 (Terminaison et Complexité)** *Si  $C_I$  est l'ensemble des clauses à un noeud donné, et si  $B_I \subseteq C_I$  est l'ensemble des clauses binaires de  $C_I$ , alors l'algorithme termine et sa complexité dans le pire des cas est en  $O(|V_{B_I}| \cdot |V_{C_I}|)$ .*

**Preuve**

*Lorsque à partir de  $C_I$ , on essaie de déduire  $x$  ( $C_I \models x$  ?), on va propager au pire  $|V_{C_I}|$  monolitéraux.*

*Si  $x$  n'est pas déductible ( $C_I \not\models x$ ) on va choisir un nouveau littéral de  $V_{B_I}$ . Si les littéraux déduits lors de l'échec  $C_I \cup \{\neg x\} \not\models \square$  n'appartiennent jamais à  $V_{B_I}$ , on va tester au pire  $|V_{B_I}|$  littéraux. D'où le résultat  $O(|V_{B_I}| \cdot |V_{C_I}|)$ .*

### 3 La méthode d'avalanche AVAL

La combinaison de l'algorithme de production de monolitéraux (RLI) avec la procédure DP donne la méthode d'énumération avalanche que nous appelons AVAL. Cette appellation est due au phénomène d'avalanche de monolitéraux que la méthode AVAL entraîne à partir d'une certaine profondeur ( $\frac{n}{21}$  en pratique) dans l'arbre de recherche lors de la résolution de problèmes aléatoires situés dans la région des problèmes difficiles. Quand il n'y a pas de monolitéral (ou de littéral pur) à instancier on appelle la procédure RLI afin de chercher un littéral impliqué par l'ensemble des clauses courant. Au retour, si il y en a un on l'affecte à vrai en le traitant comme une clause unitaire normale, sinon on choisit, moyennant une heuristique, la prochaine variable à instancier, ce qui entraîne alors un noeud (point de choix) dans l'arbre de recherche. Ainsi, en affectant un littéral impliqué par l'ensemble des clauses dès qu'il en existe un, nous minimisons le nombre de noeuds de la procédure DP (par rapport à une heuristique donnée). La méthode AVAL est décrite dans la figure 2.

L'heuristique de choix des variables propositionnelles que nous utilisons dans notre méthode est décrite dans la section suivante.

#### 3.1 Heuristiques

##### 3.1.1 Heuristique Mom

Lorsqu'il n'y a pas de monolitéral, ni de littéral impliqué par la procédure RLI (cf figure 1), il est important de choisir la "meilleure variable" à instancier, par rapport à un critère donné. Etant donné que plus le nombre de clauses binaires est important, plus la probabilité de déduire un littéral avec RLI est grande, nous avons dans un premier temps utilisé l'heuristique Mom (Maximum occurrences in minimum size clauses) (Freeman

[8]). Celle ci choisit comme prochaine variable à instancier celle qui a le plus d'occurences dans les clauses de taille minimale (de longueur 2). Formellement :

**Définition 1** Soit  $x$  un littéral. On définit  $w(x) = \sum_{-x \in C_i} 5^{-|C_i|}$  comme étant le poids de  $x$ .

L'heuristique Mom choisit comme prochaine variable à instancier celle qui maximise la fonction suivante :  $H(x) = 1024.w(x).w(\neg x) + w(x) + w(\neg x)$ .

La fonction de poids  $H$  proposée par Freeman dans [8] établit l'équilibre dans l'arbre de recherche.

### 3.1.2 Heuristique UP

L'heuristique Mom n'exploite pas entierement l'efficacité de la propagation unitaire. L'heuristique UP ([8], [11]), permet d'exploiter au mieux cette propriété et a donné de très bons résultats. Le principe en est le suivant : Soit  $C_I$  l'ensemble de clauses courant et  $x$  une variable de  $C_I$ . Soient  $C'_I = C_I \cup \{x\}$  et  $C''_I = C_I \cup \{\bar{x}\}$ . On propage les monolitéraux de  $C'_I$  et  $C''_I$ , et on instancie la variable qui réduit le plus de clauses dans  $C'_I$  et  $C''_I$ . Etant donné que cette heuristique est d'une complexité élevée, nous avons limité son utilisation aux premiers niveaux de l'arbre de recherche, là où la déduction de monolitéraux échoue souvent. Les résultats obtenus avec ces deux heuristiques sont donnés à la section 4.1.

## 3.2 Prétraitement

Il peut être intéressant, avant de commencer la recherche, d'effectuer des prétraitements qui consistent à faire des résolvantes sur la base de clauses de départ. On rend donc explicites des contraintes cachées, ce qui peut amener à réduire l'espace de recherche. Nous avons utilisé les résolvantes comme le fait Chu Min Lee dans [11]. Pour 3-SAT, nous n'effectuons que les résolvantes de taille au plus égale à 3. Les nouvelles clauses pouvant être utilisées afin de produire d'autres résolvantes. Le processus étant maintenu jusqu'à saturation. Deux contraintes sont imposées : Deux clauses binaires ne peuvent créer qu'une clause unaire. Une clause binaire et une clause ternaire ne peuvent créer qu'une clause binaire. Ce prétraitement permet un gain en temps de l'ordre de 10% pour les problèmes aléatoires situés dans la région difficile.

## 4 Expérimentations

Les expérimentations vont comprendre plusieurs parties : pour commencer, on va regarder comment se comporte notre méthode AVAL par rapport à la méthode de Davis & Putnam classique, on étudiera ensuite différentes caractéristiques de notre méthode. On terminera par la comparaison de notre méthode avec deux algorithmes parmi les meilleurs existant dans la littérature (SATZ [11], POSIT [8]) sur deux types de problèmes distincts : les instances issues des challenges DIMACS et BEIJING et les instances 3-SAT aléatoires. Les instances aléatoires sont définies de la manière suivante : Soit  $c$  le nombre de clauses et  $v$  le nombre de variables, on crée alors aléatoirement les  $c$  clauses parmi les  $2^3 \binom{v}{3}$  possibles. De nombreuses recherches ont montré qu'il existe une phase de transition pour les problèmes  $k$ -SAT aléatoires : quand on fait varier le rapport  $\frac{c}{v}$  on passe des problèmes très peu contraints (qui ont beaucoup de modèles) aux problèmes très contraints (qui n'ont pas de modèles). Ce qui fait croire à l'existence d'une valeur de seuil  $\frac{c}{v}$  pour laquelle les problèmes sont satisfaisables au dessous et inconsistant au dessus. C'est dans cette région de transition que se situent les problèmes difficiles. Pour 2-SAT le seuil a été prouvé égal à 1 (Chvátal [2]). Pour 3-SAT l'existence du seuil a été prouvée par Friedgut [9]. On ne connaît pas sa valeur exacte, mais uniquement des bornes ( $3.03 < \frac{c}{v} < 4.64$ ).

Tous les résultats expérimentaux ont été obtenus sur un PC Pentium 200 avec 64 Mo de RAM. Le code du programme est écrit en C et comprend 1300 lignes. Tous les temps CPU que nous donnons sont en secondes.

### 4.1 Comparaison entre DP et Aval

Nous avons comparé la méthode de Davis & Putnam et notre méthode AVAL de la manière suivante : nous avons utilisé les deux heuristiques décrites en 3.1 avec ces deux méthodes et nous avons généré des instances aléatoires situées au rapport  $\frac{c}{v} = 4.25$  (50 problèmes à chaque pas). Les tableaux 1a et 1b montrent

les résultats obtenus. L'ajout de l'algorithme RLI à la méthode classique de Davis & Putnam induit un gain en temps qui augmente au fur et à mesure que le nombre de variables augmente, ce qui est prometteur pour résoudre des instances de grande taille. Le nombre de noeuds (points de choix) parcourus est réduit de manière assez importante. On remarquera que plus le nombre de variables augmente, plus l'importance de l'ordre d'instanciation des variables augmente, et donc, plus l'intérêt de l'heuristique UP croît. C'est pour cette raison que nous l'utiliserons lors de la comparaison avec SATZ et POSIT.

Ainsi, lorsque l'on couple notre algorithme de recherche des littéraux impliqués (RLI) à la procédure de Davis & Putnam, on obtient un gain au niveau du nombre de noeuds, mais aussi au niveau du temps CPU. En choisissant d'affecter un monolittéral (explicite ou déduit) dès qu'il est possible de le faire nous optimisons le nombre de points de choix effectués par une procédure énumérative de type DP utilisant une heuristique donnée.

		140	160	180	200	220	240
Mom	DP	756	1165	3193	6736	18997	51733
	AVAL	69	98	245	474	1212	2672
UP+MOM	DP	654	1047	2709	5271	15373	33000
	AVAL	58	82	189	335	923	2072

**Tableau 1a:** Comparaison entre DP et AVAL (Noeuds)

		140	160	180	200	220	240
Mom	DP	0.219	0.355	1.01	2.269	7.055	19.8
	AVAL	0.176	0.273	0.717	1.554	4.508	12.1
UP+MOM	DP	0.261	0.409	1.039	2.131	6.15	14.8
	AVAL	0.228	0.339	0.79	1.535	4.50	10.5

**Tableau 1b:** Comparaison entre DP et AVAL (Tps)

## 4.2 Taux de réussite de la production de monolittéraux

Quand on regarde les résultats du tableau 2, on s'aperçoit qu'environ 90% des recherches de monolittéraux réussissent. Ceci est un résultat encourageant et explique pourquoi l'on gagne autant de noeuds par rapport à DP classique. En fait lorsque l'on expérimente la méthode AVAL, on se rend compte qu'à partir d'une certaine profondeur de l'arbre de recherche ( $\frac{|V|}{21}$  dans la pratique) on ne fait que des propagations unitaires avant de trouver un modèle ou d'arriver à l'inconsistance.

Les productions de monolittéraux qui échouent se situent principalement en haut de l'arbre. Les résultats pratiques ont montré qu'après la première recherche de monolittéraux réussie il n'y avait pratiquement plus de points de choix. Le même phénomène se passe dans la méthode DP classique sauf qu'il commence beaucoup plus tard. On peut penser que l'on a minimisé le nombre de noeuds qu'une méthode d'énumération, avec une certaine stratégie de choix de variables, doit parcourir. Statistiquement on parcourt de vrais noeuds jusqu'à une profondeur de l'ordre de  $\frac{|V|}{21}$  avec l'algorithme de l'avalanche et jusqu'à une profondeur de  $\frac{|V|}{13}$  avec DP classique. Ce qui explique la différence du nombre de noeuds entre les deux méthodes.

<i>Nb Prop</i>	100	140	180	220	260
<i>Nb appels à RLI</i>	170	860	3795	14136	62672
<i>Nb appels positifs</i>	152	784	3494	13105	58372
%	89	91	92	92	93

**Tableau 2:** % productions monolittéraux réussies

## 4.3 Efficacité de la procédure RLI

La complexité théorique de notre algorithme à chaque noeud est dans le pire des cas en  $O(|V_{B_I}| \cdot |V_{C_I}|)$  (cf proposition 4). Mais les résultats expérimentaux montrent que la complexité pratique en moyenne est linéaire

avec le nombre de variables, comme on le voit dans le tableau 3 (le rapport  $\frac{b}{a}$  montre le nombre moyen de propagation unitaire par appel de la procédure RLI). C'est cette complexité linéaire qui fait que nous avons un algorithme utilisable dans la pratique.

<i>Nb proposition</i>	100	140	180	220	260
<i>a=nb appel à RLI</i>	170	860	3795	14136	62672
<i>b=nb propagations monolit</i>	9919	73303	420363	1930100	10216322
<i>rapport <math>\frac{b}{a}</math></i>	58	85	110	136	163

**Tableau 3 :** Complexité algorithme de recherche monolitéraux

#### 4.4 Seuil de production des monolitéraux

On a effectué des tests pour savoir combien de clauses binaires (en moyenne) étaient nécessaires pour déduire un littéral (l'appel de la procédure RLI donne un résultat positif). On sait que le seuil de consistance de 2-SAT est égal à 1. Donc si l'on a autant de clauses binaires que de variables, la production d'un littéral est presque certaine de réussir. Les expérimentations sur les problèmes aléatoires ont montré que pour un rapport "nombre de clauses binaires sur nombre de propositions" aux alentours de 0.7 la production de monolittéral réussit souvent. Le tableau 4 donne la valeur moyenne du rapport  $\frac{Nb\ binaires}{Nb\ prop}$  pour laquelle l'appel de RLI est positif. Il serait intéressant de savoir s'il existe un seuil théorique de déduction de monolitéraux. Et, s'il en existe un, de connaître sa valeur.

<i>Nb Prop</i>	100	140	180	220	260	300
<i>Nb binaires</i>	0.702	0.708	0.747	0.748	0.744	0.757

**Tableau 4 :** Rapport nb binaires sur nb prop

#### 4.5 Comparaison avec SATZ et POSIT

##### 4.5.1 Problèmes aléatoires

Nous avons comparé deux méthodes connues dans la littérature (SATZ et POSIT) avec notre algorithme AVAL (utilisé avec l'heuristique UP+MOM) au niveau du temps CPU et du nombre de noeuds. Pour ce, nous avons fait varier le nombre de variables de 100 à 400 (par pas de 50) et généré des problèmes aléatoires au rapport  $\frac{c}{v} = 4.25$  (200 problèmes jusqu'à 300 variables, puis 100 problèmes). Le tableau 5 montre les résultats obtenus.

Les résultats que nous montrons sont assez encourageants. La méthode AVAL résout (en moyenne) des problèmes au seuil avec 400 variables en moins de 2 heures. Au niveau du temps CPU la méthode SATZ est la meilleure, AVAL et POSIT sont comparables. Au niveau du nombre de noeuds parcourus, notre méthode est la meilleure. L'efficacité de la méthode SATZ est due à l'utilisation de l'heuristique UP sur des variables judicieusement choisies.

		100	150	200	250	300	350	400
AVAL	<i>Tps (sec)</i>	0.069	0.268	1.681	12,1	100	610	5278
	<i>Nds</i>	14	72	382	2182	13 946	70 405	502 803
SATZ	<i>Tps (sec)</i>	0.068	0.205	0.856	4,399	30.6	189	1096
	<i>Nds</i>	18	111	590	3089	18 371	100 014	521 349
POSIT	<i>Tps (sec)</i>	0.016	0.148	1.074	8.605	65.7	407	3698
	<i>Nds</i>	39	264	1502	10094	65 505	334 847	2 898 510

**Tableau 5 :** Les problèmes aléatoires

##### 4.5.2 Challenges DIMACS et BEIJING

Nous avons également comparé ces trois algorithmes sur les problèmes issus des challenges DIMACS et BEIJING. Nous avons limité à deux heures (7200 secondes) le temps maximum qu'un algorithme doit mettre

pour résoudre une instance. Les problèmes du challenge BEIJING sont listés individuellement, alors que l'on a regroupé par classe ceux du challenge DIMACS. *Tps* désigne alors le temps global pour résoudre l'ensemble des problèmes d'une classe (lorsqu'un problème d'une classe n'est pas résolu en 2 heures, le temps crédité est alors de 2 heures),  $\#M$  dénote le nombre de problèmes de la classe et  $\#S$  le nombre de problèmes résolus. (nous avons supprimé les instances par32, f, g et hanoi5 qu'aucune des trois méthodes ne résolvent). Les tableaux 6 et 7 nous montrent les résultats obtenus.

Les problèmes issus du challenge BEIJING sont des problèmes d'ordonnancement, de planning et de synthèse qui présentent un grand nombre de symétries. Nous résolvons une instance de plus que SATZ et 4 de plus que POSIT. Hormis 2bitadd\_12, nos temps sont comparables ou meilleurs que ceux de POSIT et comparables à ceux de SATZ.

Voyons maintenant les problèmes issus du challenges DIMACS. Nous résolvons moins d'instances que SATZ pour les classes dubois, ii16 et ssa, et moins d'instances que POSIT pour la classe ii16. Nous sommes les seuls à résoudre entièrement la classe ii32, et POSIT résout seulement la moitié des instances de aim-200. Ces résultats montrent que l'apport de la technique de recherche des littéraux impliqués est significatif pour ces instances.

<i>Problème</i>	AVAL		SATZ		POSIT	
	<i>Tps</i>	<i>Nds</i>	<i>Tps</i>	<i>Nds</i>	<i>Tps</i>	<i>Nds</i>
2bitadd_10	3706	2 116 944	>7200	-	>7200	-
2bitadd_11	6.4	4 838	113	120 982	>7200	-
2bitadd_12	35.2	28 843	0.265	99	0.04	35
2bitcomp_5	0.02	11	0.01	6	0.01	34
2bitmax_6	0.06	14	0.05	7	0.05	12
3bitadd_31	>7200	-	>7200	-	>7200	-
3bitadd_32	>7200	-	3101	297 652	>7200	-
3blocks	2.2	16	1.6	7	2.38	669
4blocks	1184	18 823	930	228 040	>7200	-
4blocksb	11.5	97	8	8	70	8424
e0ddr2-10-by-5-1	2657	705	86	35	>7200	-
e0ddr2-10-by-5-4	617	661	86	32	2726	34759
enddr2-10-by-5-1	38	10	>7200	-	>7200	-
enddr2-10-by-5-8	66	15	81	30	>7200	-
ewddr2-10-by-5-1	41	15	124	40	143	250
ewddr2-10-by-5-8	861	238	92	39	>7200	-

**Tableau 6 :** Challenge Beijing

<i>Classe Pb</i>	$\#M$	AVAL		SATZ		POSIT	
		$\#S$	<i>Tps</i>	$\#S$	<i>Tps</i>	$\#S$	<i>Tps</i>
aim-50	24	24	14	24	14	24	0.3
aim-100	24	24	3.55	24	3.4	24	320
aim-200	24	24	4.2	24	3.85	12	86400
dubois	13	8	43274	12	38665	8	45500
hole	5	5	180	5	213	5	444
ii8	14	14	15.4	14	5	14	0.77
ii16	10	8	14967	10	104	9	7268
ii32	17	17	2060	16	7638	15	14410
jnh	50	50	12.84	50	11	50	0.25
par8	10	10	0.7	10	0.66	10	0.05
par16	10	10	288	10	403	10	33
ssa	8	7	10500	8	826	7	7231

**Tableau 7 :** Challenge DIMACS

## 5 Conclusion

Les méthodes complètes du type Davis et Putnam utilisent la propagation des monolitéraux. Or, cette propagation est effectuée uniquement sur les monolitéraux apparents. La méthode d'avalanche AVAL utilise, en plus des monolitéraux existant dans la base de clauses, des propagations sur des monolitéraux impliqués par l'algorithme RLI, de déduction de clauses unitaires que nous avons mis au point. Cet algorithme consiste à produire des littéraux appartenant à des clauses binaires avec de bonnes propriétés de complexité, dans la pratique ( $O(|V|)$ ). La combinaison de DP avec L'algorithme RLI constitue la méthode AVAL.

Grace à cet algorithme, nous avons mis en évidence une structure de l'arbre de recherche dans le cas des problèmes aléatoires situés au seuil. En effet, deux parties se distinguent dans l'arbre de recherche. Jusqu'à une profondeur  $\frac{|V|}{21}$  qui délimite, dans la pratique, la première partie, l'algorithme effectue la totalité des vrais noeuds et donc la partie difficile. Au delà de cette profondeur le phénomène d'avalanche de monolitéraux se déclenche et la méthode achève la recherche en temps polynomial, c'est la partie de l'arbre dite "facile".

L'expérimentation a montré l'intérêt de la méthode AVAL. Son point fort est l'implication des monolitéraux cachés qui permet, par rapport à une heuristique donnée, de minimiser le nombre de points de choix de l'arbre de recherche.

Les comparaisons effectués avec les algorithmes SATZ et POSIT montrent l'efficacité de notre méthode, surtout au niveau du nombre de noeuds, même si la méthode SATZ de par l'optimalité de son heuristique est meilleure en temps CPU.

## Références

- [1] H. Levesque B. Selman and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence AAAI'94*, 1994.
- [2] V. Chvátal and B. Reed. Mick Gets Some (the odds are on this side). In *33rd IEEE Symposium on Foundation of Computers Science*, 1992.
- [3] V. Chvátal and E. Szemerédi. Many Hard Examples for Resolution. *journal of ACM*, 1988.
- [4] S. Cook. The Complexity of Theorem Proving Procedures. In *Third Annual ACM Sump. On Th. of Computing*, 1971.
- [5] J. Crawford and L. Auton. Experimental Results on the Crossover Point in Random 3sat. In *proceedings of the Eleventh National Conference on Artificial Intelligence(AAAI-93)*, 1993.
- [6] M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 1960.
- [7] O. Dubois and Y. Boufkhad. A General Upper Bound for the Satisfiability Threshold of random r-sat formulae. *Journal of Algorithms*, 1996.
- [8] J.W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Univ. of Pennsylvania, Philadelphia, 1995.
- [9] E. Friedgut. Necessary and sufficient conditions for sharp thresholds of graphs properties and the k-sat problem. Technical report, Institute of Mathematics, The Hebrew University of Jerusalem, 1997.
- [10] Chu Min Li and Anbulagan. Look-Ahead Versus Look-Back for Satisfiability Problems. *Lecture Notes in Computer Science*, 1996.
- [11] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problem. In *IJCAI 97*, 1997.
- [12] W.V. Quine. Methods of logics. *Henry Holt, New York*, 1950.
- [13] H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the 14th International Conference on Automated deduction*, 1997.