

Cet article est paru dans *les actes de la deuxième  
Conférence Nationale sur la Résolution Pratique de  
Problèmes NP-Complets*, Dijon, FRANCE, 1996.

## Deux approches pour la résolution du problème SAT \*

**Mazure Bertrand**

CRIL  
Université d'Artois  
rue de l'Université SP 16  
F-62307 Lens Cedex  
France  
*mazure@lens.lifl.fr*

**Saïs Lakhdar**

CRIL - IUT de Lens  
Université d'Artois  
rue de l'Université SP 16  
F-62307 Lens Cedex  
France  
*sais@lens.lifl.fr*

**Grégoire Eric**

CRIL  
Université d'Artois  
rue de l'Université SP 16  
F-62307 Lens Cedex  
France  
*gregoire@lens.lifl.fr*

### Résumé

Dans ce papier, nous présentons une double utilisation d'une méthode de recherche locale pour le problème SAT. Nous montrons comment l'utilisation et la taille d'une liste tabou influent considérablement les performances des algorithmes de recherche locale. De plus nous décrivons comment cette méthode peut être exploitée pour prouver l'inconsistance.

### Mots Clés

SAT, NP-complet, méthodes de recherche locale, inconsistance locale

---

\* Ce travail est soutenu par le projet Ganymède II du Contrat de Plan Etat/Nord - Pas de Calais, par le PRC-IA "Aspects Algorithmiques de la résolution des problèmes exprimés à l'aide de contraintes", par le Ministère de l'Enseignement Supérieur et de la Recherche et par l'Institut Universitaire de Technologie de Lens.

# 1 Introduction

Le problème SAT -Satisfiabilité d'une formule booléenne mise sous forme normale conjonctive- est un problème fondamental, à la fois théoriquement et pratiquement. En effet il occupe un rôle central en théorie de la complexité, où il représente le problème NP-complet de référence [2], alors que de nombreux problèmes en informatique s'y ramènent naturellement ou le contiennent.

Récemment, la communauté a constaté un regain d'intérêt dans la compréhension de la nature de la difficulté du problème SAT [1, 6]. Dans un même temps, plusieurs auteurs proposaient de nouveaux algorithmes ou heuristiques -simples mais efficaces- permettant d'élargir considérablement la classe des problèmes traitables en pratique [7, 15, 16].

En effet, une classe de problèmes difficiles, les instances K-SAT, fut caractérisée ([1, 6, 13]). Afin d'adresser ces problèmes, deux familles d'algorithmes ont été isolées. La première est constituée des techniques incomplètes basées sur le principe des réparations locales. La seconde est formée des techniques complètes du type Davis & Putnam [4].

Dans ce papier, nous présentons une technique basée sur le principe de la recherche locale, TWSAT (pour "Taboo Walk Strategy for SAT") et nous analyserons les apports de cette technique dans l'une et l'autre des deux familles.

Dans la prochaine section, nous décrirons l'algorithme TWSAT, basée sur l'utilisation systématique d'une liste de réparations interdites (appelée liste tabou). Puis nous présenterons un résultat expérimental obtenu sur la longueur de la liste tabou lors de la recherche d'optimisation de cette méthode sur les instances K-SAT. Afin de valider notre algorithme, nous comparerons les performances de TWSAT avec celles de GSAT utilisant Random Walk Strategy de B. Selman *et al.* [16]. Ensuite nous définirons les notions d'inconsistance *globale* et *locale* et nous analyserons dans quel contexte les méthodes de recherche locale peuvent être employées dans le but de détecter l'inconsistance. Nous proposerons ensuite un schéma de combinaison entre une méthode complète : Davis & Putnam [4] et une méthode incomplète : TWSAT [11] et illustrerons les performances de cette combinaison sur de nombreux problèmes [5].

**Procédure** *TWSAT*

**Entrée :** *un ensemble de clauses  $S$ , la longueur de la liste tabou  $l$ , le nombre de flips  $MAX-FLIPS$ , et le nombre d'essais  $MAX-TRIES$*

**Sortie :** *une affectation des variables propositionnelles satisfaisant  $S$ , si celle-ci est trouvée ; rien sinon*

**Début**

**pour**  $i := 1$  **jusque**  $MAX-TRIES$

$I :=$  *une interprétation aléatoire de l'ensemble des variables propositionnelles*

**pour**  $j := 1$  **jusque**  $MAX-FLIPS$

**si**  $I$  *satisfait*  $S$  **alors retourner**  $I$

**sinon**

$x :=$  *la variable n'appartenant pas à la liste tabou dont l'inversion minimise le nombre de clauses falsifiées*

$liste\_tabou[j \text{ modulo } l] := x$

$I := I$  *avec la valeur de  $x$  inversée*

**fin si**

**fin pour**

**fin pour**

**Fin**

Figure 1: l'algorithme TWSAT

## 2 TWSAT : l'algorithme

TWSAT s'inscrit dans la famille des algorithmes de réparations locales -type GSAT [15]-. Cette méthode s'appuie sur l'ossature GSAT de base et maintient à jour une liste de réparations interdites afin d'éviter les réparations locales récurrentes et d'échapper aux minima locaux.

Précisément, TWSAT gère une file de longueur fixe, des variables "flippées" -cf. Figure 1-. Après chaque flip, la variable ayant été réparée entre dans cette liste d'interdits où elle y restera durant un certain nombre d'itérations correspondant à la longueur de la liste. Ceci permet d'explorer et de couvrir au mieux l'espace de recherche.

Il faut noter que TWSAT n'est pas un nouvel algorithme, c'est le Tabou traditionnel [9] sans utilisation du critère d'aspiration. Ce qui signifie que la liste d'interdits est utilisée tout au long de la recherche et non pas, seulement, lorsqu'une détérioration de la configuration courante doit être effectuée. C'est-à-dire, tant qu'une variable est dans la liste tabou elle ne peut participer à la réparation même si son inversion provoque l'évolution de la configuration dans le sens de la progression la plus forte.

Le paramétrage des méthodes à recherche locale joue un rôle fondamental sur leur efficacité. En effet, Selman *et al.* [16] propose un réglage pointu des paramètres de leur méthode : *MAX-FLIPS*, *MAX-TRIES* et surtout la probabilité  $p$  -qui influe énormément sur les performances de son algorithme-. Il en est de même pour TWSAT où le réglage de la longueur de la liste d'interdits  $l$  est primordial.

### 3 Réglage de TWSAT : résultats expérimentaux

Afin de déterminer la taille optimale de la liste tabou, nous avons effectué des séries d'expérimentations. Nous nous sommes particulièrement intéressés au réglage de TWSAT pour les instances K-SAT. Nous avons généré des instances 3-SAT au seuil dont le nombre de variables varie entre 50 et 1500 par pas successifs de 50 variables. Pour chaque taille de problèmes nous avons testé un nombre significatif d'instances (500 pour les problèmes de 50 à 1000 variables, 100 pour les problèmes de taille supérieure à 1000).

Pour chaque instance, nous avons expérimenté TWSAT avec une longueur de liste tabou variant de 1 à 50. La Figure 2 montre la longueur optimale de la liste tabou en fonction du nombre de variables.

Sur l'intervalle considéré la courbe apparait comme une fonction linéaire du nombre de variables. De plus, nous avons également constaté :

- que plus la longueur de la liste tabou s'éloigne de la longueur optimale, plus les performances de TWSAT se dégradent.
- que cette longueur optimale reste identique pour des instances générées en dehors du seuil.

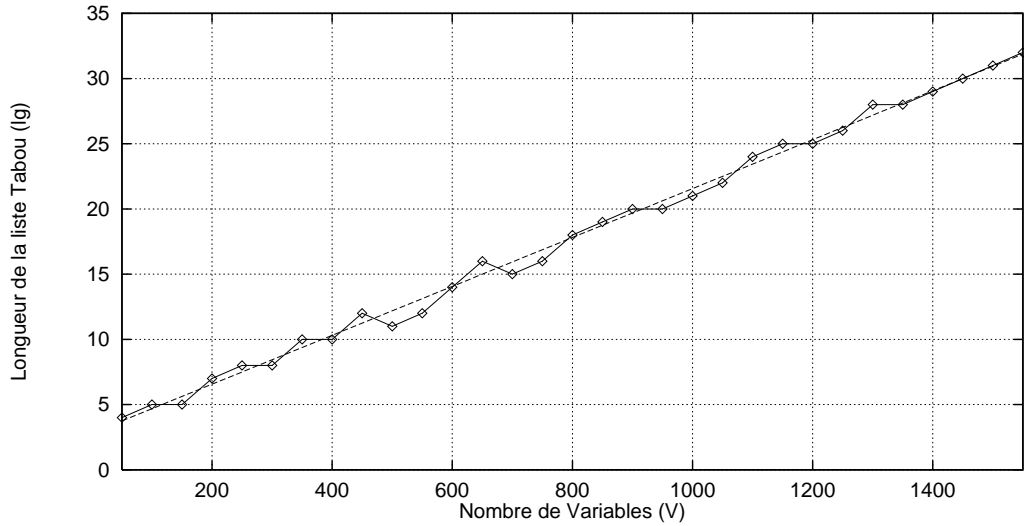


Figure 2: Taille optimale de la liste tabou pour des instances 3-SAT ( $C/V=4.3$ )

Equation de la longueur optimale observée :  $lg = 0.01875 V + 2.8125$

- que la taille tabou semble ne pas dépendre de la taille des clauses -i.e.  $K$ -, ceci a été observé en reconduisant le même type d'expérimentations, à moindre échelle (100 instances pour un nombre de variables variant entre 50 et 600 par pas successifs de 50 variables), sur les instances 4-SAT.

Il nous semble que cette linéarité pourrait être liée à la taille des optima locaux - i.e. au nombre nécessaire de dégradations successives de la configuration courante pour s'échapper d'un optimum local -. Cependant ceci reste une conjecture à prouver à la fois expérimentalement (en déterminant une relation entre la taille des instances et la taille moyenne des optima locaux de ces instances) et théoriquement.

## 4 GSAT vs TWSAT

Dans cette section, nous présentons une comparaison expérimentale entre GSAT et TWSAT sur des instances 3-SAT au seuil. Pour ce faire les deux algorithmes ont été implantés sur une plate-forme<sup>1</sup> commune, écrite en C sous Linux 1.1.59.

Les algorithmes ont été comparés respectivement sur plusieurs critères :

- le nombre de réparations effectuées en moyenne
- le pourcentage de problèmes résolus<sup>2</sup>
- le temps d'exécution moyen

Nous avons également indiqué le rapport du nombre de réparations sur le pourcentage de problèmes résolus. Ceci afin d'obtenir une comparaison réaliste lorsqu'un algorithme résout plus d'instances que l'autre.

Les réglages des paramètres (MAX-TRIES & MAX-FLIPS) utilisés par GSAT et TWSAT sont ceux fournis par B. Selman *et al.* dans [16], la probabilité utilisée pour GSAT est de 50{que la longueur de la liste d'interdits pour TWSAT est fournie par l'équation donnée précédemment. De plus, pour chaque taille de problèmes, 500 instances ont été générées, à l'exception des instances à 2000 variables où seules 50 instances ont été testées.

La Table 1 synthétise les résultats obtenus et montrent que TWSAT est plus efficace que GSAT.

---

<sup>1</sup>Cette plate-forme est disponible et peut-être obtenue, soit en la sollicitant par mail, à l'adresse suivante : mazure@lens.lifl.fr, soit par ftp anonyme : ftp.lifl.fr, répertoire pub:projects/SAT

<sup>2</sup>Le pourcentage de problèmes résolus est relatif au 50% de problèmes estimés satisfiables au seuil.

problèmes		Nb. inst.	GSAT			
n	c		temps (sc.)	nb flips	résolus	ratio
100	430	500	.18	2803	88%	31.85
200	860	500	1.99	18626	73%	255.85
400	1700	500	15.03	204670	100%	2046.70
600	2550	500	19.59	250464	62%	4013.85
800	3400	500	140.61	1809986	67%	26854.39
1000	4250	500	369.88	4633763	57%	81009.84
2000	8240	50	3147.26	26542387	16%	1658899.19

problèmes		Nb. inst.	TWSAT			
n	c		temps (sc.)	nb flips	résolus	ratio
100	430	500	.11	1633	93%	17.60
200	860	500	.73	9678	74%	130.78
400	1700	500	11.51	145710	100%	1457.10
600	2550	500	13.92	167236	65%	2580.80
800	3400	500	99.45	1143444	71%	16150.34
1000	4250	500	292.10	3232463	62%	51802.29
2000	8240	50	3269.15	29415465	40%	735386.63

Table 1: GSAT vs. TWSAT<sup>3</sup>

Il faut également remarquer que la différence positive en faveur de TWSAT s'accroît lorsque le nombre de variables augmente. Cela est illustré par la courbe suivante (cf. Figure 3).

Nous expérimentons actuellement TWSAT sur d'autres modèles de générations aléatoires, ainsi que sur des problèmes réels présentés au Challenge DIMACS [5]. Les résultats préliminaires montrent un gain considérable sur les instances aléatoires générées par les modèles de génération (mixed clause-length model) proposés par I.P. Gent et T. Walsh [8].

Dans les prochaines sections, nous analyserons dans quelles mesures il est possible d'allier les avantages des techniques complètes et incomplètes afin de prouver plus efficacement l'inconsistance et plus particulièrement les inconsis-

---

<sup>3</sup>Les temps exprimés dans cette table correspondent au temps moyen nécessaire à l'un ou l'autre des algorithmes pour exhiber un modèle. Ces temps ont été obtenus sur des PC de type i486 DX2 66.

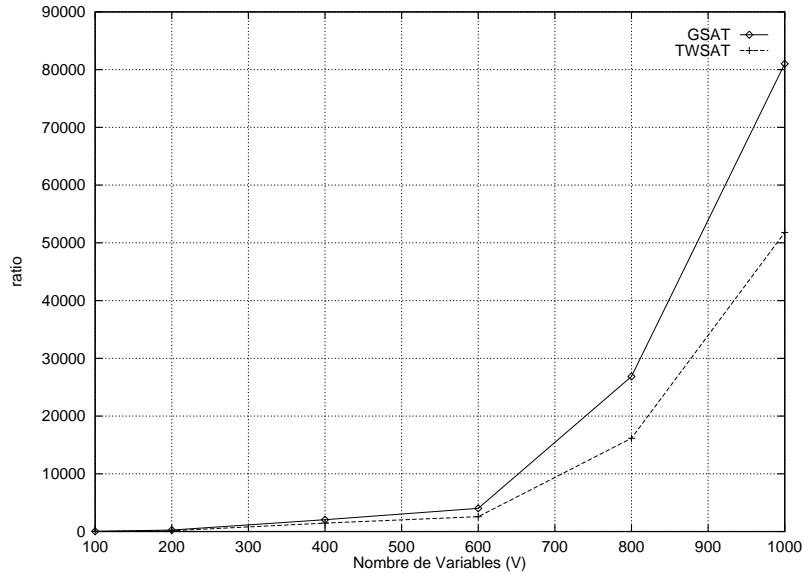


Figure 3: Résultats de GSAT et TWSAT pour 3-SAT au seuil

tances locales que nous définissons ci-dessous.

## 5 Inconsistance locale vs Inconsistance globale

Un problème fondamental en représentation des connaissances et en démonstration automatique réside dans la difficulté d'adresser et de prouver les différentes formes d'inconsistance.

Nous commençons par une définition du concept d'inconsistance globale et locale.

### Définition 1

Une instance SAT  $S$  est dite *globalement inconsistante*<sup>4</sup>  
ssi  
 $S$  est inconsistante et  $\forall S' \subset S : S'$  est consistante.

---

<sup>4</sup>on dit également *minimalement inconsistante*

Lorsqu'une instance SAT inconsistante n'est pas inconsistante globalement, elle est dite localement inconsistante.

### Définition 2

Une instance SAT  $S$  est dite *localement inconsistante*  
ssi  
 $S$  est inconsistante et  $\exists S' \subset S : S'$  est inconsistante

Clairement, plusieurs degrés de localité peuvent être définis et peuvent, par exemple, être caractérisés par le rapport de la taille du noyau inconsistant sur la taille de l'instance. Ces formes d'inconsistance sont d'une importance capitale dans de nombreuses applications, très fréquemment l'inconsistance d'une base de connaissances est due à la présence accidentelle d'une information contradictoire sur un sujet donné. Cette inconsistance pollue globalement l'ensemble de la base de connaissances (i.e. toute information ainsi que son contraire peuvent être inférés de cette base). Malheureusement, il n'existe pas de méthode universelle et efficace pour détecter et/ou localiser ces inconsistances.

Il est évident que les problèmes inconsistants globalement sont les plus difficiles à prouver ; mais heureusement ce sont généralement des problèmes générés de manière artificielle et qui ne se retrouvent pas dans les applications réelles. De plus ce genre d'instances possède le plus souvent de très fortes régularités (problème des "pigeons" [3], formules de Tseitin et Urquhart [17], etc.).

## 6 Utilisation de TWSAT pour circonscrire l'inconsistance

Nous allons expliciter dans cette section comment une méthode de type recherche locale, comme TWSAT, peut être utilisée pour localiser l'inconsistance d'instances SAT, sachant que de telles méthodes ne sont destinées qu'à la recherche de solutions. La question que nous nous posons est de savoir dans quelle mesure il est possible d'exploiter le travail effectué par ce type de méthodes pour aller plus loin dans la preuve de l'inconsistance.

Afin de répondre à cette question nous avons effectué une série de tests sur des problèmes inconsistants et avons analysé le comportement de TWSAT sur

ces instances. Lors de ces exécutions nous avons totalisé pour chaque littéral, son nombre d'apparitions dans une clause falsifiée et pour chaque clause, le nombre de fois qu'elle a été falsifiée. Il nous semble vraisemblable que les plus falsifiées ont une forte probabilité de constituer le noyau inconsistant, de même les littéraux étant apparus le plus souvent dans les clauses falsifiées ont une forte chance d'appartenir à l'ensemble de variables du noyau inconsistant.

Cette hypothèse s'est avérée expérimentalement correcte. En effet, lorsque l'on tente de résoudre un problème localement inconsistant au moyen de méthodes de réparation locale, comme TWSAT, la trace (décrite ci-dessus) permet de scinder le problème en deux parties : une consistante et une inconsistante [12].

Les résultats obtenus montrent qu'une méthode ne tenant pas compte de cette notion de localité d'inconsistance, pourra être amenée à résoudre la partie inconsistante du problème autant de fois qu'il existe de solutions pour la partie consistante. Cette trace de TWSAT nous semble donc importante et nous allons montrer comment l'utiliser dans le cadre d'un algorithme mixte.

## 7 DP+TWSAT une méthode mixte

Nous allons décrire comment la trace fournie par TWSAT peut être exploitée par des méthodes complètes, comme la procédure Davis & Putnam [4].

L'algorithme que nous avons mis au point DP+TWSAT [12] est un algorithme complet se basant sur la procédure DP et utilisant TWSAT à la fois comme heuristique (choix de la prochaine variable à affecter) et comme un moyen de prolonger l'interprétation partielle effectuée par DP vers un modèle. Plus précisément, à chaque point de choix de l'arbre de recherche effectué par DP, nous appliquons TWSAT sur le problème simplifié par l'interprétation courante. Deux cas se présentent au retour de l'appel de TWSAT :

1. TWSAT a exhibé un modèle du problème simplifié, dans ce cas l'algorithme s'arrête, le problème est alors satisfiable (l'interprétation partielle est alors prolongée par le modèle exhibé par TWSAT).
2. TWSAT a échoué dans la résolution du problème simplifié, dans ce cas le littéral étant apparu le plus souvent dans les clauses falsifiées lors de

```

Procédure DP + TWSAT
Entrée : un ensemble de clauses S
Sortie : la consistance de S
Début
  Unit_propagation(S);
  si il existe une clause vide alors retourner (0);
  sinon si toutes les variables sont affectées alors retourner (1)
  sinon si TWSAT( S ) trouve un modèle alors retourner (1)
  sinon
    p := le littéral étant apparu le plus souvent dans les
      clauses falsifiées durant la recherche de TWSAT;
    return (DP+TWSAT( S∧p) ∨ DP+TWSAT(S∧¬p));
  fin si;
Fin

```

Figure 4: la procédure DP+TWSAT

la recherche de TWSAT est élu par DP comme prochaine variable à affecter.

Cet algorithme est détaillé dans la Figure 4.

Afin de vérifier la faisabilité et la validité d'un tel algorithme, il nous est nécessaire de comparer cette approche avec un DP utilisant une heuristique plus traditionnel.

## 8 FFIS vs TWSAT

Dans cette section, nous montrons les résultats obtenus par la méthode mixte, décrite précédemment, sur de nombreux problèmes et comparons ces résultats avec ceux obtenus par DP utilisant l'heuristique "First Fail In Shortened" [14], variante de l'heuristique Jeroslow-Wang [10].

Notons que notre objectif principal au travers de ces expérimentations est de tester la faisabilité de notre méthode, c'est-à-dire analyser l'influence des variables fournies par TWSAT sur la taille de l'arbre de recherche de DP.

Pour ce faire, nous n'avons pas cherché d'optimisations particulières de cet algorithme, nous décrirons par la suite quelques optimisations possibles.

Nous avons comparé les deux algorithmes sur la plupart des instances fournies à DIMACS [5].

La Table 2 donne un échantillon significatif de nos expérimentations, elle montre, en outre, les améliorations incontestables obtenues par cette approche. Nos objectifs ont été atteints, non seulement sur les problèmes inconsistants, mais également sur certains problèmes consistants non solvables à la fois par les méthodes incomplètes et complètes. De plus, pour les problèmes consistants nous conservons l'efficacité des méthodes incomplètes puisqu'elles sont appliquées tout au long de la recherche.

En outre ces résultats, nous avons essayé d'isoler un noyau minimale-ment inconsistant -i.e. inconsistant globalement- pour chacun des problèmes (plusieurs noyaux peuvent exister pour un même problème). Une constatation est que la taille des noyaux isolés est très petite par rapport à la taille initiale du problème. La Table 3 nous donne quelques exemples de cette réduction.

Notons que les problèmes "duboisX" proposés par O. Dubois [5] sont des problèmes très difficiles et inconsistants globalement. Les résultats obtenus (en terme de nombre d'affectations) sur ces problèmes montrent l'influence des littéraux retournés par TWSAT sur la taille de l'arbre de recherche de DP.

## 9 Optimisations naturelles

Il faut noter que ces tests n'ont été conduits que pour vérifier la faisabilité d'une technique de recherche mixte. C'est pour cela que nous n'avons pas cherché à optimiser notre méthode. Nous pensons accentuer considérablement les performances d'abord en effectuant un réglage pointu des paramètres de la méthode à recherche locale et en déterminant un compromis optimal entre le temps passé par DP et par TWSAT. Plus précisément, nous proposons les optimisations naturelles suivantes :

- Dans l'algorithme décrit précédemment, nous choisissons juste le littéral apparu le plus souvent dans les clauses falsifiées lors de la recherche de TWSAT, ceci afin de guider DP. De plus, nous faisons appel à TWSAT

---

<sup>5</sup>Une étoile (\*) signifie que le problème n'a pas pu être résolu au bout de 30 heures de temps CPU.

DIMACS instances	Sat.	DP+ffis			DP+TWSAT		
		nb affect.	nb choix	temps	nb affect.	nb choix	temps
aim-50-1_6-yes1-2	Oui	8980	586	0s29	17	4	0s08
aim-50-1_6-no-3	Non	54614	4527	1s56	92	12	0s32
aim-50-2_0-yes1-3	Oui	7616	446	0s28	23	3	0s09
aim-50-2_0-no-1	Non	54014	2759	1s84	80	8	0s25
aim-100-1_6-yes1-2	Oui	495858	30052	16s86	127	10	0s43
aim-100-1_6-no-3	Non	67786124	3854371	2221s17	176	17	0s78
aim-100-2_0-yes1-3	Oui	361660	18153	20s40	234	14	0s95
aim-100-2_0-no-1	Non	38503642	1725544	1527s69	46	5	0s27
aim-200-1_6-yes1-3	Oui	*	*	*	216	10	0s98
aim-200-1_6-no-1	Non	*	*	*	244	17	1s92
aim-200-2_0-yes1-4	Oui	*	*	*	299	27	3s59
aim-200-2_0-no-4	Non	*	*	*	176	18	1s74
bf0432-007	Non	938037830	6208925	86086s93	153760	1079	615s09
bf1355-075	Non	317628	2047	68s97	13606	67	169s87
bf1355-638	Non	*	*	*	54890	259	584s23
bf2670-001	Non	*	*	*	275844	66869	39126s23
ssa0432-003	Non	133794	1570	7s96	2944	33	5s79
ssa2670-130	Non	*	*	*	14188726	66869	39126s23
ssa2670-141	Non	294818640	2312147	27814s59	14845888	99794	25471s94
ssa7552-159	Oui	1557	84	0s82	12	1	0s83
ssa7552-160	Oui	1457	76	0s72	1	1	1s20
dubois10	Non	20564	2063	0s51	11624	1031	3s03
dubois11	Non	41044	4111	1s01	28752	2375	7s24
dubois12	Non	82004	8207	1s98	47428	3485	12s71
dubois13	Non	163924	16399	3s93	96312	6839	25s37
dubois14	Non	327764	32783	7s89	112920	8479	35s38
dubois15	Non	655444	65551	15s90	186652	14321	60s43
dubois16	Non	1310804	131087	32s31	374972	28223	135s07
dubois17	Non	2621524	262159	64s95	916636	73735	308s97

Table 2: DIMACS<sup>5</sup>

DIMACS instances	Taille initiale		Taille du noyau	
	Nb. variables	Nb. clauses	Nb. variables	Nb. clauses
aim-200-2_0-no-4	200	400	36	42
bf2670-001	1393	3434	79	139
ssa0432-003	435	1027	306	322
dubois17	51	136	51	136

Table 3: Taille des noyaux inconsistants

à chaque fois que DP doit choisir un littéral à affecter. A l’opposé, nous pourrions faire qu’un seul appel à TWSAT et utiliser les littéraux dans l’ordre décroissant de leurs apparitions dans les clauses falsifiées pour guider DP à chaque point de choix. Entre ces deux points de vue extrêmes, nous pensons qu’une attitude optimale peut être trouvée en limitant le nombre d’appels aux méthodes de recherche locale. Ce compromis dépend de la forme de la trace donnée par la méthode de recherche locale et de la position du point de choix dans l’arbre de recherche de DP.

- Une seconde optimisation possible a trait au réglage des paramètres de la méthode de recherche locale employée. En effet la taille et la nature des problèmes fournis à la méthode de recherche locale diffèrent à chaque appel. De ce fait, une adaptation dynamique des paramètres de cette méthode s’impose afin d’en tirer les meilleures performances.

## 10 Conclusion

De ce papier, deux résultats émergent. Le premier montre que le réglage des paramètres ainsi que la variation dans l’utilisation des stratégies de réparations jouent un rôle crucial sur les performances des algorithmes de recherche locale (utilisation systématique d’une liste tabou, réglage de la taille de cette liste, ...), ceci rend difficile et discutable la comparaison de ces techniques. Le second montre que la coopération entre méthode systématique et non systématique est une voie de recherche intéressante.

Actuellement nous développons une méthode mixant les deux types d’approches d’une autre manière. Celle-ci exploite la trace des clauses et non pas celle des littéraux.

Un prolongement naturel de ce travail sera d’expliquer théoriquement la linéarité de la taille optimale de la liste tabou - existe-t-il un lien entre cette taille et la taille moyenne des optima locaux ? - et d’essayer de caractériser formellement les littéraux retournés par la méthode de recherche locale. Il serait également intéressant de voir dans quelles mesures ces résultats peuvent être étendus à la résolution du problème MAX-SAT.

## References

- [1] V. Chvátal, E. Szemerédi (1988). Many Hard Examples for Resolution. *Journ. of the ACM*, vol. 33, no. 4, pp. 759-768.
- [2] S. Cook (1971). The complexity of theorem proving procedures *Third Annual ACM Symp. On Th. of Computing*, pp. 151-158.
- [3] S. Cook (1976). A Short Proof of the Pigeon Hole Principle Using Extended Resolution. *SIGACT News*, vol. 8, pp. 28-32.
- [4] M. Davis, H. Putnam (1960). A Computing Procedure for Quantification Theory. *Journ. of the ACM*, vol. 7, pp. 201-215.
- [5] DIMACS (1993). Proc. du second challenge organisé par “the Center for Discrete Mathematics and Computer Science of Rutgers University” en 1993.
- [6] O. Dubois, J. Carlier (1991). Probabilistic Approach to the Satisfiability Problem. *Theoretical Computer Science*, vol. 81, pp. 65-75.
- [7] O. Dubois, P. André, Y. Boufkhad, J. Carlier (1993). SAT versus UNSAT. *Journ. of the Am. Mathematical Society* (submitted). also in *Proc. of the Second DIMACS Challenge*.
- [8] I.P. Gent, T. Walsh (1994). The SAT Phase Transition. *Proc. ECAI-94*, pp. 105-109.
- [9] F. Glover (1989) Tabu search - Part I. *ORSA Journ. of Computing*, vol. 1, pp. 190-206.
- [10] R.E. Jeroslow, J. Wang (1990). Solving Propositional Satisfiability Problems. *Ann. Math. AI*, vol 1, pp. 167-187.
- [11] B. Mazure B., L. Saïs, E. Grégoire (1995). TWSAT: a New Local Search Algorithm for SAT. Performance and Analysis. *Proc. CP-95 Workshop on Studying and Solving Really Hard Problems*, pp. 127-130.

- [12] B. Mazure, L. Saïs, E. Grégoire E. (1996). Detecting Logical Inconsistencies. *Proc. AI/MATH-96*, pp. 116-121.
- [13] D. Mitchell, B. Selman, H. Levesque (1992). Hard and Easy Distributions of SAT Problems. *Proc. AAAI-92*, pp. 459-465.
- [14] A. Rauzy (1994). On the Complexity of the Davis and Putnam's Procedure on Some Polynomial Sub-Classes of SAT. *LaBRI Technical Report 806-94*, Université de Bordeaux 1.
- [15] B. Selman, H. Levesque, D. Mitchell (1992). A New Method for Solving Hard Satisfiability Problems. *Proc. AAAI-92*, pp. 440-446.
- [16] B. Selman, H.A. Kautz, B. Cohen (1993). Local Search Strategies for Satisfiability Testing. *Proc. 1993 DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*.
- [17] G.S. Tseitin (1968). On the Complexity of Derivations in Propositional Calculus. in: Slisenko A.O. (ed.), *Structures in Constructive Mathematics*, Part II, pp. 115-125.