

GASAT : une approche hybride pour le problème SAT

Frédéric Lardeux, Frédéric Saubion et Jin-Kao Hao

LERIA, Université d'Angers,

2 Bd Lavoisier,

F-49045 Angers Cedex 01

email: {lardeux, saubion, hao}@info.univ-angers.fr

Résumé

Ce papier présente GASAT, un algorithme hybride pour le problème de satisfiabilité (SAT). Un croisement spécifique génère des configurations qui sont améliorées par une recherche tabou possédant des mécanismes de sélection et de diversification adaptés au problème SAT. Les performances de GASAT sont évaluées sur des benchmarks bien connus. Les comparaisons avec les algorithmes de l'état de l'art pour le problème SAT montrent que GASAT donne des résultats très compétitifs.

1 Introduction

Le problème de satisfiabilité (SAT)[Garey et Johnson, 1978] consiste à trouver une affectation booléenne validant une formule en logique propositionnelle. Une instance de ce problème est définie par un ensemble de variables booléennes (aussi appelées atomes) $\mathcal{X} = \{x_1, \dots, x_n\}$ et une formule booléenne $\phi : \{0, 1\}^n \rightarrow \{0, 1\}$. Un littéral est une variable ou sa négation; une clause est une disjonction de littéraux. Une formule est en forme normale conjonctive (CNF) si c'est une conjonction de clauses. Une affectation est une fonction $v : \mathcal{X} \rightarrow \{0, 1\}$. La formule est dite satisfiable s'il existe une affectation rendant ϕ vraie et est non satisfiable s'il n'en existe pas. Une formule est vraie si et seulement si toutes ses clauses sont vraies.

SAT est un des six principaux problèmes NP-complet et a de nombreuses applications telles que les tests et vérifications de circuits intégrés (VLSI), le diagnostic de pannes, les emplois du temps ... Sa résolution est généralement considéré comme un problème de décision mais il peut aussi être abordé comme :

- recherche de solution : trouver une ou plusieurs solutions
- MAX-SAT : trouver une affectation qui satisfasse le plus de clauses possibles
- comptage : calculer le nombre de solutions

Les algorithmes dédiés au problème SAT se divisent en deux classes : les algorithmes complets et les algorithmes incomplets.

Les algorithmes complets répondent au problème de décision et sont principalement basés sur la procédure Davis-Putnam-Loveland [Davis *et al.*, 1962]. Ils diffèrent essentiellement dans la règle de branchement utilisée pour explorer l'arbre de recherche [Castell *et al.*, 1996; Dubois *et al.*, 1996; Li et Anbulagan, 1997; Zhang, 1997; Mazure *et al.*, 1998]. Des techniques spécifiques telles que la détection de symétries, de backbones ou encore d'équivalences permettent de renforcer quelques algorithmes [Benhamou et Sais, 1992; Li, 2000; Dubois et Dequen, 2001].

Les algorithmes incomplets sont généralement basés sur la recherche locale [Hansen et Jaumard, 1990; Spears, 1996; Huang et Jin, 1997] et sur les algorithmes évolutionnaires ou hybrides [De Jong et Spears, 1989; Hao et Dorne, 1994; Fleurent et Ferland, 1994; Jaumard *et al.*, 1994; Eiben *et al.*, 1998; Gottlieb *et al.*, 2002; Habet *et al.*, 2002]. GSAT [Selman *et al.*, 1992], le premier algorithme de descente, et Walksat [Selman *et al.*, 1994], sa version améliorée, sont les exemples les plus connus d'algorithmes incomplets. Bien que les algorithmes incomplets ne soient pas d'un grand secours pour le problème de décision dans le cas des instances insatisfiables, ils sont très utiles pour le problème de décision dans le cas des instances satisfiables, ils sont les seuls à pouvoir trouver des solutions pour les très grandes instances.

Incités par le challenge n° 6 de [Selman *et al.*, 1997], nous nous intéressons ici au développement d'une approche incomplète s'appuyant sur la structure des instances. Nous avons choisi pour cela d'intégrer dans notre algorithme GASAT une recherche tabou (RT) dans un contexte évolutionnaire basé sur un opérateur de recombinaison. Alors que la RT assure une intensification de la recherche de solution dans certaines zones, la recombinaison de configurations permet d'assurer une diversification de cette recherche. C'est donc dans l'optique de garantir un équilibre entre intensification et diversification, et d'obtenir ainsi une exploration efficace de l'espace de recherche, que nous nous sommes tournés vers ce type de combinaison qui a déjà montré sa puissance pour de nombreux problèmes tels que la coloration de graphes. La clé de voûte d'une telle hybridation est bien sûr la définition d'un opérateur de recombinaison spécifique qui prendra en compte les relations structurelles entre les variables induites par les clauses. Les croisements spécifiques pour le problème SAT sont très peu étudiés [Fleurent et Ferland, 1994]. D'un certain côté, GASAT présente des similitudes avec l'algorithme proposé dans [Fleurent et Ferland, 1994]. GASAT se distingue par son croisement spécialisé, la puissance de sa RT et l'interaction entre ces deux opérateurs.

Dans la section suivante, nous présenterons l'algorithme GASAT. Nous analyserons ensuite le croisement ainsi que la RT. Nous finirons par comparer GASAT à d'autres algorithmes afin d'en évaluer les performances.

2 L'Algorithme Génétique Hybride : GASAT

Dans cette section nous présentons l'algorithme GASAT et les notions de bases relatives aux algorithmes évolutionnaires. Un algorithme évolutionnaire [Holland, 1975] a pour but de faire évoluer une population d'individus pour les adapter à un critère de qualité. Cette évolution s'effectue au moyen d'opérateurs tels que le croisement ou la mutation. L'ajout d'une recherche locale sur ces opérateurs permet d'obtenir un algorithme hybride alternant des phases d'intensification avec des phases de diversification. Cette combinaison a déjà montré sa puissance pour de nombreux problèmes.

2.1 Représentation d'un individu

La manière la plus évidente de représenter un individu pour une instance SAT avec n variables est une chaîne de n bits où chaque variable est associée à un bit. Un individu correspond donc à une affectation. C'est cette représentation qui est utilisée dans GASAT. D'autres représentations sont possibles comme le montre [Gottlieb *et al.*, 2002]. Dans la représentation que nous utilisons, l'espace de recherche est l'ensemble $\mathcal{S} = \{0, 1\}^n$ (c'est à dire toutes les chaînes de n bits possibles). Si X est un individu alors $X[i]$ indique la valeur de la $i^{\text{ième}}$ variable et $X[[i \leftarrow \alpha]$ est un individu X dont la $i^{\text{ième}}$ variable prend la valeur α . La fonction booléenne $sat(X, c)$ nous indique si l'affectation associée à l'individu X rend vraie la clause c .

2.2 Fonction d'évaluation

Soient ϕ une formule CNF et X un individu, l'évaluation de X est le nombre de clauses fausses engendrées par l'affectation associée à X :

$$eval: \mathcal{S} \rightarrow \mathbb{N}$$

$$eval(X) = |\{c | \neg sat(X, c) \wedge c \in \phi\}|$$

où $|E|$ indique la cardinalité de l'ensemble E . Cette fonction d'évaluation induit un ordre $>_{eval}$ sur les individus de la population. Cette fonction a comme borne supérieure le nombre de clauses de la formule étudiée et comme borne inférieure 0. Tout individu ayant une évaluation de 0 est une solution.

2.3 L'algorithme GASAT

L'algorithme final (figure 1) est obtenu en combinant un mécanisme de croisement et une RT. Pour une population donnée, dans laquelle chaque individu représente une affectation, la première étape consiste à sélectionner les meilleurs individus par rapport à $>_{eval}$. Deux parents sont ensuite choisis parmi ces individus et l'opérateur de croisement leur est appliqué. Le fils ainsi obtenu est amélioré par la RT puis intégré à la population initiale sous certaines conditions. Ce processus est réitéré jusqu'à ce qu'une solution soit trouvée ou que le nombre de croisements autorisés soit atteint.

3 Composants Spécifiques

Dans cette section les opérateurs de croisement et de RT sont présentés ainsi que quelques fonctions qui leur sont utiles.

3.1 Fonctions utiles

Dans les deux opérateurs principaux de GASAT, des fonctions communes sont utilisées : la fonction *flip* et la fonction *amélioration*.

La fonction *flip* permet d'inverser la valeur d'une variable pour un individu donné.

$$flip: \{0, 1\} \rightarrow \{0, 1\}$$

$$flip(x) = 1 - x$$

La fonction *amélioration* calcule l'amélioration d'un individu lorsqu'une de ses variables est flipée. Elle permet d'évaluer les améliorations possibles.

$$amélioration: \mathcal{S} \times \mathbb{N} \rightarrow \mathbb{N}$$

Données: une formule \mathcal{F} en CNF, $Maxflip$, $Maxessais$
Résultat: la meilleure solution trouvée

Début
CréationPopulation(P)
 $essai \leftarrow 0$
Tant qu'aucun $X \in P$ ne valide ϕ et que $essai < Maxessais$
/* Sélection */
 $P' \leftarrow Selection(P, n)$
Choix de $X, Y \in P'$
/* Croisement */
 $Z \leftarrow croisement(X, Y)$
/* Amélioration par RT */
 $Z \leftarrow RT(Z, Maxflip)$
/* Condition d'insertion pour le fils */
Si $\forall X \in P, Z >_{eval} X$ alors le plus vieux $X \in P$ est remplacé par Z
 $essai \leftarrow essai + 1$
FinTantQue
S'il existe $X \in P$ validant ϕ
alors l'affectation correspondant est retournée
sinon la meilleure affectation trouvée est retournée
Fin

FIG. 1 – L'algorithme GASAT

$$amélioration(X, i) = |\{c \mid sat(X[i \leftarrow flip(X|i)], c) \wedge \neg sat(X, c)\}| - |\{c \mid \neg sat(X[i \leftarrow flip(X|i)], c) \wedge sat(X, c)\}|$$

Cette fonction¹ a déjà été utilisée dans GSAT et Walksat [Selman *et al.*, 1992; 1994]. Remarquons que le résultat peut être négatif ce qui indique que l'individu est dégradé. Cette fonction induit un ordre $>_{amélioration}$ sur les variables qui peut être étendu aux individus ($X >_{amélioration} Y$ ssi il existe une position i telle que $\forall j, X|i >_{amélioration} Y|j$).

3.2 Croisement

Un croisement efficace doit prendre en compte la sémantique de l'individu afin de contrôler le processus de recherche. Pour le problème SAT, les individus sont représentés par des variables binaires contraintes par les clauses. Afin d'obtenir un croisement efficace pour SAT, il faut prendre en compte la structure induite par les clauses.

L'objectif de notre croisement est de construire un fils qui corrige les clauses fausses identiques chez les deux parents et qui préserve au maximum les clauses vraies. Pour ce faire, nous prenons l'intersection de l'ensemble des clauses fausses des deux parents et nous les rendons vraies chez le fils. Ensuite nous conservons les variables ayant des valeurs identiques chez les deux parents afin de conserver au maximum la structure des parents. A ce point du croisement nous n'avons plus, chez le fils, que des clauses vraies ou indéterminées car certaines variables

1. Par la suite nous noterons cette fonction $amélioration(X|i)$

ne sont pas encore évaluées. Le fait d'affecter des valeurs aléatoirement à ces variables peut faire apparaître des clauses fausses. Cela n'empêche pas le croisement de corriger les clauses fausses identiques chez les deux parents et de conserver une certaine structure venant des parents.

Définition 1 : Corrective Clause Crossover (CC crossover)

Soient X et Y les deux parents et Z le fils. Pour chaque clause c telle que $\neg \text{sat}(X, c) \wedge \neg \text{sat}(Y, c)$ et pour toutes les positions i telles que la variable x_i apparaisse dans c et que x_i ne soit pas évaluée, nous calculons $\sigma_i = \text{amélioration}(X|i) + \text{amélioration}(Y|i)$ et nous fixons $Z|k = \text{flip}(T|k)^2$ où k est la position telle que σ_k est maximum et $T = X$ si $\text{amélioration}(X|i) > \text{amélioration}(Y|i)$, sinon $T = Y$. Toutes les variables non encore évaluées prennent la valeur d'un des deux parents avec la même probabilité.

Exemple

L'exemple suivant illustre le CC crossover. Le problème possède cinq variables, $\{x_1, x_2, x_3, x_4, x_5\}$ et sept clauses:

$$(x_1 \vee x_3 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee \neg x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee \neg x_5) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4 \vee x_5) \wedge (\neg x_2 \vee x_3 \vee x_4)$$

Soit X et Y deux individus ayant chacun deux clauses fausses:

	x_1	x_2	x_3	x_4	x_5
X	1	1	0	0	1
Y	0	1	0	0	1

- 1) La seconde et la dernière clause sont fausses pour X et Y . Nous calculons donc σ pour tous les x_i .

Pour la seconde clause : $\sigma_2 = 3, \sigma_3 = 4, \sigma_5 = 2$. Nous fixons donc x_3 à 1.

Pour la dernière clause : $\sigma_2 = 3, \sigma_4 = 4$. Nous fixons donc x_4 à 1.

	x_1	x_2	x_3	x_4	x_5
Z			1	1	

- 2) Les variables x_1, x_2 et x_5 prennent chacune leur valeur aléatoirement chez X ou Y .

	x_1	x_2	x_3	x_4	x_5
Z	0	1	1	1	1

3.3 Recherche Tabou (RT)

L'opérateur de recherche locale de GASAT est une RT [Glover et Laguna, 1997]. Cette méthode utilise une mémoire afin d'éviter de stagner dans les optima locaux. Elle agit comme une descente (à chaque itération, elle fait le meilleur mouvement); une fois visitée, une configuration est rendue taboue. L'algorithme n'est pas autorisé à la revisiter avant un nombre donné d'itérations. La mémorisation de la dernière configuration étant très coûteuse, une alternative consiste à rendre tabou le mouvement ayant mené à cette configuration (ce qui est plus contraignant). Un mouvement est donc accepté s'il est le meilleur et s'il n'est pas tabou. Une fois exécuté, le mouvement est mis dans la liste tabou qui agit comme une file de taille λ et de type

2. On peut remarquer que si une clause est fautive pour les deux parents, alors toutes les variables apparaissant dans cette clause ont nécessairement la même valeur chez les deux parents. Cela vient du fait qu'une clause est fautive ssi tous ses littéraux sont faux.

FIFO. Le mouvement sort de la file après λ itérations.

Dans notre contexte, il est clair que les mouvements sont les flips des variables d'un individu et que la liste *tabou* contient les flips déjà effectués. Cette procédure est résumée figure 2. Ici la configuration initiale est remplacée par le fils obtenu par le croisement. Le fait d'empêcher des mouvements interdit plus de configurations que nécessaire. Par conséquent, dans le cas où un flip *tabou* améliore quand même la meilleure solution obtenue, il est accepté. C'est le phénomène d'aspiration.

```

RT:  $S \times IN \times IN \rightarrow S$ 
RT( $Z, \lambda, Maxflip$ ) =
Début
Génération d'une configuration initiale  $Z$ 
Création d'une liste tabu de taille  $\lambda$ 
Meilleure  $\leftarrow Z$ 
nbflip  $\leftarrow 0$ 
Tant Que not ( $eval(Meilleure) = 0 \vee nbflip > Maxflip$ ) faire
  Choisir  $i$  tel que amélioration( $Z|i$ ) est maximale (*)
  Si  $i \notin tabu$  alors
    Si  $eval(Z|i \leftarrow flip(Z|i)) < eval(Meilleure)$  alors
      Meilleure  $\leftarrow Z|i \leftarrow flip(Z|i)$ 
    FinSi
  Sinon /* Aspiration */
    Si  $eval(Z|i \leftarrow flip(Z|i)) < eval(Meilleure)$  alors
      Meilleure  $\leftarrow Z|i \leftarrow flip(Z|i)$ 
    Sinon
      Recommencer le choix (*) avec comme condition
      supplémentaire que  $i$  soit différent du  $i$  actuel
    FinSi
  FinSi
   $Z \leftarrow Z|i \leftarrow flip(Z|i)$ 
  nbflip  $\leftarrow nbflip + 1$ 
  Sortir la plus vieille variable de tabu
  Ajouter  $i$  dans tabu
FinTantQue
Retourner Meilleure
Fin

```

FIG. 2 – Recherche *tabou* (RT)

3.4 Spécialisation de la RT

Deux mécanismes ont été ajoutés à notre RT afin d'affiner la sélection de la variable à flipper et de créer plus de diversité quand l'algorithme est en difficulté.

3.4.1 Affinage du choix de la variable à flipper

Lors du choix de la variable à flipper, il est assez courant que plusieurs variables soient candidates. Dans la RT standard, une d'entre elles est choisie aléatoirement. Afin de diminuer l'ensemble des choix possibles, nous avons ajouté un critère supplémentaire à cette sélection.

Constatant que plus une clause avait un nombre important de littéraux vraies, plus il était facile de flipper une de ses variables sans rendre cette clause fausse, nous introduisons la notion de degré de vérité:

$$degré(c) = |\{x|x = 1, x \in c\}|$$

Les variables sélectionnées pour un éventuel flip sont donc celles où la fonction suivante est maximale :

$$poids(i) = \sum_{valeur=0}^1 \frac{(2 \times valeur - 1) \sum_{c \in \{x|x \in clauses, lit(i) \in x, lit(i)=valeur\}} degré(c)}{|\{x|x \in clauses, lit(i) \in x, lit(i) = valeur\}|}$$

où *clauses* représente l'ensemble des clauses de la formule et *lit(i)* correspond au littéral associé à la variable *i*. Si toutefois, plusieurs variables sont encore candidates, une d'entre elles est choisie aléatoirement.

3.4.2 Diversification

Au cours de la RT, il est fréquent que le nombre de clauses fausses devienne rapidement très faible mais n'atteigne pas zéro. Ces dernières clauses fausses sont souvent les mêmes et mettent en échec la RT qui vient buter sur elles. Nous appelons ces clauses des *clauses-butoir*.

Pour éviter ce problème, un nouveau mécanisme de diversification a été développé. Lorsqu'une clause apparait un certain nombre de fois comme clause-butoir, elle est rendue vraie de manière forcée par le flip d'une de ses variables puis les clauses fausses engendrées par ce flip sont elles même rendues vraie et ainsi de suite *k* fois (empiriquement fixé à 5). Les variables flippées sont mises dans une liste plus forte que la liste tabou n'autorisant pas le phénomène d'aspiration. Cela oblige ces clauses à rester vraies un certain temps.

Ce mécanisme permet de sortir des optima locaux plus facilement qu'avec une RT classique comme le montrent les pics de la figure 3.

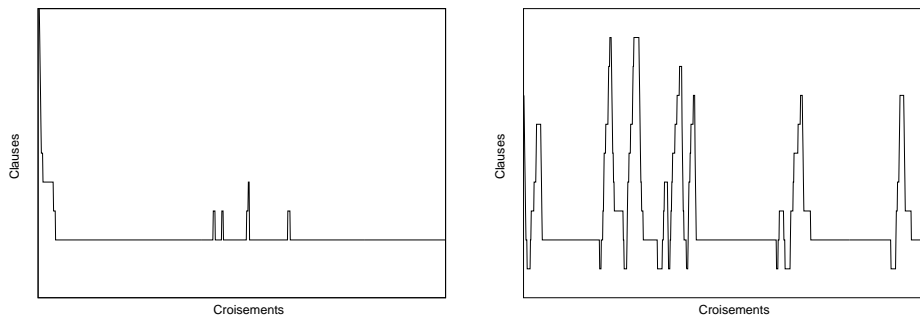


FIG. 3 – RT sans diversité (à gauche) et méthode avec diversité (à droite)

4 Études comparatives

Afin d'évaluer la puissance des opérateurs de GASAT, nous les avons comparées avec des opérateurs existants. Nous étudions dans une première partie le croisement puis dans une seconde la RT.

4.1 Étude comparative du croisement

Pour étudier les avantages du CC crossover, nous l'avons comparé à trois autres croisements connus.

4.1.1 Croisements seuls

Définition 2 : Croisement uniforme

La valeur de chaque variable α du fils est choisie aléatoirement parmi la valeur de α de chacun des deux parents.

Définition 3 : Croisement de type multi-points

Soient X et Y les deux parents et Z le fils. Nous calculons $m = \max(\text{amélioration}(X|i), \text{amélioration}(Y|i))_{i \in [1..n]}$. Si $X >_{\text{amélioration}} Y$ alors $\forall i, \text{amélioration}(X|i) > \frac{m}{2} \Rightarrow Z|i = \text{flip}(X|i)$ et $\forall i, \text{amélioration}(X|i) \leq \frac{m}{2} \Rightarrow Z|i = Y|i$.

Définition 4 : Croisement de Fleurent et Ferland (F&F) [Fleurent et Ferland, 1994]

Soient X et Y les deux parents et Z le fils. Pour chaque clause c telle que $\text{sat}(X, c) \wedge \neg \text{sat}(Y, c)$ (resp. $\neg \text{sat}(X, c) \wedge \text{sat}(Y, c)$) et pour toutes les positions i telles que la variable x_i apparaisse dans c , $Z|i = X|i$ (resp. $Z|i = Y|i$). Toutes les variables non encore évaluées prennent la valeur d'un des deux parents avec la même probabilité.

Pour que les quatre croisements soient étudiés dans les mêmes conditions, la sélection des parents se fait aléatoirement dans une population de 100 individus et le fils ainsi obtenu remplace l'individu le plus ancien de la population. Les tests ont été réalisés sur un benchmark 3-sat aléatoire généré au seuil, f500.cnf [Mitchell *et al.*, 1992]. Les critères de comparaisons pour ces croisements sont le nombre moyen et le nombre minimum de clauses fausses obtenues ainsi que la diversité des populations générée. Cette diversité est donnée par la formule de l'entropie.

Définition 5 : Entropie [Fleurent et Ferland, 1994]

Soit n_{ij} le nombre de fois où la variable i a pour valeur j dans la population P .

$$\text{entropie}(P) = \frac{\sum_{i=1}^n \sum_{j=0}^1 \frac{n_{ij}}{|P|} \log \frac{n_{ij}}{|P|}}{n \log 2}$$

Les résultats obtenus figure 4 montrent que le croisement uniforme et que le croisement multi-points ne sont pas efficaces. Le croisement F&F semble efficace du point de vue de l'

amélioration du nombre moyen de clauses fausses de la population comme du meilleur individu. Le CC crossover paraît moins performant de ce point de vue qualitatif mais la diversité des fils générés est plus intéressante. La population gérée par le CC crossover sera de bonne qualité avec des individus assez différents alors qu'une population gérée par le croisement F&F serait de très bonne qualité mais avec une diversité très faible.

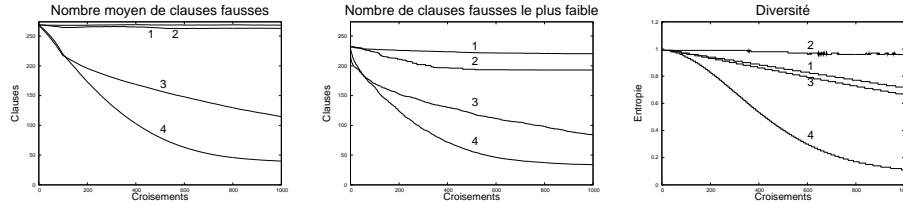


FIG. 4 – Comparaisons des croisements avec sélection aléatoire des parents et insertion de tous les enfants (1→uniforme, 2→multi-points, 3→CC crossover et 4→F&F)

4.1.2 Influence des opérateurs spécifiques

GASAT étant un algorithme hybride, il combine plusieurs mécanismes. Des opérateurs de sélection de parents et d'insertion des fils ont été ajoutés pour améliorer encore les performances du croisement.

Mécanisme de sélection des parents: Les deux parents sont sélectionnés aléatoirement dans un ensemble \mathcal{P} . Cet ensemble est composé des k (ici 15) meilleurs individus de la population ayant tous au moins une clause fausse différente deux à deux.

Condition d'insertion du fils: La condition pour qu'un fils X soit inséré dans la population est: $\exists Z \in \mathcal{P}$ tel que $eval(Z) < eval(X)$. Si cette condition est remplie alors X remplace l'individu le plus vieux de la population, sinon X est détruit.

Les résultats présentés figure 5 montrent que l'ajout des ces deux opérateurs modifie le comportement des croisements précédents. Pour le croisement multi-points et le croisement F&F, ces opérateurs dégradent les performances. Par contre, pour le croisement uniforme et le CC crossover, les résultats sont meilleurs. Le croisement uniforme ne donne toujours pas de bons résultats mais par contre, le CC crossover donne de très bons résultats en ayant une diversité plus élevée que le croisement uniforme et le croisement de F&F.

4.2 Étude de la RT

Les mécanismes de spécialisation présentés section 3.4 ralentissent l'algorithme mais permettent d'améliorer les résultats pour les benchmarks structurés (tableau 1). Les tests ont été réalisés sur la RT de GASAT. La longueur de la liste tabou a été fixée à 10% du nombre de variables du benchmark, le nombre de flips autorisés est limité à 10^5 et le nombre d'exécution est de 20. Les benchmarks utilisés sont le $\mathbb{f}500$, une instance 3-sat générée aléatoirement au seuil et le $\text{par}8-1-c$, une instance représentant un problème de parité de fonction. Les résultats sont

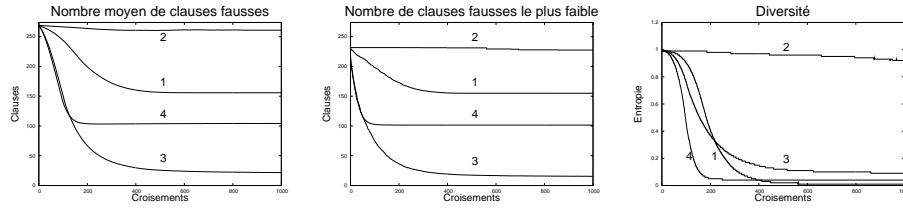


FIG. 5 – Comparaisons des croisements avec critère de sélection des parents et insertion seulement des meilleurs enfants (1→uniforme, 2→multi-points, 3→CC crossover et 4→Fleurent et Ferland)

donnés sous la forme de trois valeurs. La première est le pourcentage de réussite (%) qui est le nombre de succès divisé par le nombre d'essais, la seconde est le temps moyen en seconde (sec.) pour obtenir un succès et la troisième est le nombre moyen de flips (fl.) pour obtenir un succès.

Méthode utilisée	Instances					
	f500			par8-1-c		
	%	sec.	fl.	%	sec.	fl.
RT standard	10	0.95	54018	15	0.04	11332
RT + Affinage du choix	5	4.36	61383	35	0.02	1673
RT + Diversification	25	0.51	27940	35	0.03	8781
RT + Affinage du choix + Diversification	0	-	-	50	0.03	2951

TAB. 1 – Tabou

On peut remarquer que le mécanisme de diversification (section 3.4.2) est efficace quel que soit le type de benchmarks alors que le mécanisme d'affinage (section 3.4.1) du choix de la variable à flipper n'est efficace que pour les instances structurées. Ceci peut s'expliquer par le fait que les instances générées aléatoirement sont plus homogènes dans leurs contraintes et donc que le gain produit par le choix affiné est beaucoup trop faible pour compenser son coût calculatoire.

5 Résultats Expérimentaux

Dans cette section, nous évaluons les performances de GASAT sur différentes classes de benchmarks satisfiables ou non. Nous comparons GASAT avec Walksat [Selman *et al.*, 1994] et UnitWalk [Hirsch et Kojevnikov, 2001], le meilleur des algorithmes incomplets présentés à la compétition SAT2002 [Simon *et al.*, 2002].

5.1 Conditions d'expérimentation

Du fait de la nature incomplète et non déterministe de GASAT, Walksat et UnitWalk, chaque algorithme a été testé 20 fois sur chaque benchmarks avec un temps d'exécution limité à 1 heure sur un Sun Fire 880 (4 CPU UltraSPARC III 750 Mhz, 8 Go de RAM). Les algorithmes sont utilisés avec leur paramètres standards. Le nombre de flips est fixé à 101×10^5 pour les trois

algorithmes comparés. Nous donnons par la suite les paramétrages précis de chaque algorithme.

GASAT : GASAT travaille avec une population de 10^2 individus. Lors de la génération de cette population, une RT de 10^3 flips est appliquée à chaque individu. Le nombre de parents possibles pour le CC crossover est fixé à 15 avec comme condition qu'ils soient tous différents. Le nombre de croisements autorisé est de 10^3 et sur chaque fils une RT de 10^4 flips maximum est appliquée. La taille de la liste tabou est de 10% du nombre de variables du problème. Le mécanisme de diversification est appliqué avec une profondeur de diversification de 5 à tous les benchmarks alors que le mécanisme de sélection n'est actif que pour les instances structurées.

Walksat : L'algorithme Walksat utilisé est la version v39. Le nombre d'essais autorisé est de 10 avec 101×10^4 flips maximum par essai. Walksat utilise l'heuristique "best" avec un bruit de 0.5.

UnitWalk : La version de l'algorithme UnitWalk est : 0.98. Le nombre maximum de flips autorisé est 101×10^5 et le nombre d'essais est 1.

5.2 Benchmarks et critères d'évaluation

Deux classes d'instances ont été utilisées : les instances structurées et les instances aléatoires.

- les instances structurées :
 - `mat25.shuffled`, `mat26.shuffled` (multiplication de deux matrices $n \times n$ utilisant m produits [Li *et al.*, 2002]),
 - `color-15-4`, `color-22-5` (problèmes de coloration d'échiquier [Hao *et al.*, 2003]),
 - `difp_19_0_arr_rcr`, `difp_19_99_arr_rcr` (problèmes difficiles de factorisation d'entiers),
 - `g125.18`, `g250.29` (problèmes de coloration de graphe).
- les instances aléatoires :
 - `f1000`, `f2000` (instances DIMACS [Mitchell *et al.*, 1992]),
 - 2 instances de 500 variables générées par `hgen2` avec les graines 1205525430 et 512100147,
 - 2 instances générées avec `glassy` dont une avec 399 variables et la graine 1069116088 et une autre de 450 variables et la graine 325799114,
 - 2 instances générées avec `okgen` dont une avec 500 variables, 1000 clauses et la graine 241702789-241702789 et une autre de 700 variables, 1400 clauses et la graine 1413056357-1413056357 (instances présentées à la compétition SAT2002).

Les instances de coloration d'échiquier sont nouvelles. Elles correspondent au problème proposé dans [Hao *et al.*, 2003]. Le but est de colorier toutes les cases d'un échiquier avec k couleurs telles que les quatre coins de tous les rectangles contenus dans l'échiquier n'aient pas la même couleur.

Trois critères de comparaison sont utilisés pour évaluer GASAT et le comparer avec UnitWalk et Walksat. Le premier est le pourcentage de réussite (%) qui est le nombre de succès divisé par le nombre d'essais. Ce critère est important car il souligne la puissance de recherche de l'algorithme. Les deux autres critères sont le temps moyen en seconde (sec.) et le nombre moyen de flips (fl.) pour les succès.

5.3 Analyse des résultats

Le tableau 2 et le tableau 3 montrent respectivement les résultats de GASAT, Walksat et UnitWalk sur des instances structurées et des instances aléatoires.

Benchmarks				GASAT			Walksat			UnitWalk		
instances	var	cls	sat	%	sec.	fl.	%	sec.	fl.	%	sec.	fl.
mat25.shuffled	588	1968	N		(3 clauses)		(3 clauses)			(10 clauses)		
mat26.shuffled	744	2464	N		(2 clauses)		(2 clauses)			(15 clauses)		
color-10-3	300	6475	Y	100	208.33	929366	(2 clauses)			(12 clauses)		
color-22-5	2420	272129	?		(10 clauses)		(40 clauses)			(52 clauses)		
difp_19_0_arr_rcr	1201	6563	Y		(8 clauses)		(28 clauses)			(22 clauses)		
difp_19_99_arr_rcr	1201	6563	Y		(11 clauses)		(22 clauses)			(21 clauses)		
g125.18	2250	70163	Y	50	1878.43	149023	(10 clauses)			(24 clauses)		
g250.29	7250	454622	Y		(13 clauses)		(34 clauses)			(56 clauses)		

TAB. 2 – Instances structurées (si aucune affectation n’est trouvée, le nombre moyen de clauses fausses obtenu est donné entre parenthèses)

Benchmarks					GASAT			Walksat			UnitWalk		
gen	graine	var	cls	sat	%	sec.	fl.	%	sec.	fl.	%	sec.	fl.
glassy	1069116088	399	1862	Y		(5 clauses)		(5 clauses)			(18 clauses)		
glassy	325799114	450	2100	Y		(8 clauses)		(8 clauses)			(26 clauses)		
f1000	-	1000	4250	Y	75	69.82	843371	100	9.73	461017	100	1.04	168355
f2000	-	2000	8500	Y	15	200.79	1616845	75	23.57	511457	100	17.17	1480441
hgen2	1205525430	500	1750	Y	5	113.07	2033273	(1 clauses)			(15 clauses)		
hgen2	512100147	500	1750	Y		(1 clauses)		(1 clauses)			(21 clauses)		
okgen	241702789	500	1000	Y	100	0.02	488	100	0.01	133	100	0.01	123
okgen	1413056357	700	1400	Y	100	0.03	660	100	0.01	182	100	0.01	177

TAB. 3 – Instances aléatoires (si aucune affectation n’est trouvée, le nombre moyen de clauses fausses obtenu est donné entre parenthèses)

Ces résultats ne font pas apparaître une supériorité claire de l’un ou l’autre des algorithmes. GASAT semble globalement être plus performant sur les instances structurées que sur les instances aléatoires. Ces résultats ne sont pas surprenant car le CC crossover permet de propager les caractéristiques des structures. GASAT est particulièrement efficace sur les instances de grandes tailles par rapport à Walksat et UnitWalk. Par contre, bien que GASAT donne des résultats compétitifs sur les instances *glassy* et *hgen2*, il est moins performant sur les instances *f1000* et *f2000*. Il est intéressant de remarquer que pour le problème MAX-SAT, et ce que les instances soient structurées ou aléatoires, GASAT donne de très bons résultats. Dans le cas où aucune solution n’est trouvée et où le nombre moyen de clauses fausses obtenu est indiqué, nous ne faisons pas apparaître la variance car elle est généralement très faible quel que soit l’algorithme.

Remarques

Nous ne comparons pas GASAT avec des algorithmes complets car la taille des benchmarks utilisés est souvent trop élevée pour que ces derniers puissent donner des résultats compétitifs.

Une autre raison est que nous nous intéressons plutôt au problème MAX-SAT qui n'est généralement pas traité par les algorithmes complets.

En fonction de la famille de benchmarks (coloration, factorisation, . . .), il est possible d'affiner les paramètres afin d'obtenir des résultats encore plus intéressants. Les paramètres proposés dans ce papier sont des paramètres permettant d'obtenir de bons résultats quel que soit le benchmark étudié. Il est intéressant de remarquer qu'un paramétrage plus fin permet de résoudre l'instance `color-10-3` avec 100% de réussite en 2 secondes et 47891 flips.

6 Conclusion

Nous avons présenté l'algorithme GASAT, un algorithme génétique hybride pour le problème SAT (et MAX-SAT). Il est composé d'un opérateur de croisement corrigeant et conservant la structure des clauses et d'une RT à laquelle des mécanismes spécifiques au problème SAT ont été ajoutés. Ces deux opérateurs sont complémentaires car le croisement permet d'explorer l'espace de recherche et la RT permet d'en exploiter une zone. De plus, des mécanismes permettent d'assurer une certaine diversité dans la population.

GASAT a été évalué sur des instances aléatoires et structurées. Il a été comparé à deux algorithmes de l'état de l'art: Walksat et UnitWalk. Les résultats obtenus sont très compétitifs. GASAT donne des résultats très intéressants sur les instances de grandes tailles et apparaît comme très efficace pour le problème MAX-SAT. Par contre, il semble moins performant pour certaines instances aléatoires.

Nos prochains travaux sur GASAT vont consister à mieux comprendre son comportement en fonction des différentes classes de benchmarks et à améliorer le CC crossover en le rendant dynamique.

Remerciements : Nous tenons à remercier les relecteurs pour leurs remarques et commentaires. Ce travail est en partie financé par le projet CPER COM.

Références

- [Benhamou et Sais, 1992] Belaid Benhamou et Lakhdar Sais. Theoretical study of symmetries in propositional calculus and applications. Dans *CADE'92*, pages 281–294, 1992.
- [Castell *et al.*, 1996] Thierry Castell, Claudette Cayrol, Michel Cayrol, et Daniel Le Berre. Using the davis and putnam procedure for an efficient computation of preferred models. Dans *ECAI '96; 12th European Conference on Artificial Intelligence*, pages 350–354, August 1996.
- [Davis *et al.*, 1962] Martin Davis, George Logemann, et Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, Jul 1962.
- [De Jong et Spears, 1989] Kenneth A. De Jong et William M. Spears. Using genetic algorithm to solve NP-complete problems. Dans *Proc. of the Third Int. Conf. on Genetic Algorithms*, pages 124–132, San Mateo, CA, 1989.
- [Dubois *et al.*, 1996] Olivier Dubois, Pascal André, Yacine Boufkhad, et Jacques Carlier. SAT versus UNSAT. Dans *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 415–436, 1996.

- [Dubois et Dequen, 2001] Olivier Dubois et Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. Dans Bernhard Nebel, editor, *Proc. of the IJCAI'01*, pages 248–253, San Francisco, CA, 2001.
- [Eiben *et al.*, 1998] Agoston E. Eiben, Jan K. van der Hauw, et Jano I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.
- [Fleurent et Ferland, 1994] Charles Fleurent et Jacques A. Ferland. Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. Dans *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 619–652, 1994.
- [Garey et Johnson, 1978] Michael R. Garey et David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1978.
- [Glover et Laguna, 1997] Fred Glover et Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
- [Gottlieb *et al.*, 2002] Jens Gottlieb, Elena Marchiori, et Claudio Rossi. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation*, 10(1):35–50, 2002.
- [Habet *et al.*, 2002] Djamel Habet, Chu Min Li, Laure Devendeville, et Michel Vasquez. A Hybrid Approach for SAT. Dans *ICCP; International Conference on Constraint Programming (CP), LNCS*, 2002.
- [Hansen et Jaumard, 1990] Pierre Hansen et Brigitte Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44(4):279–303, 1990.
- [Hao *et al.*, 2003] Jin-Kao Hao, Frédéric Lardeux, et Frédéric Saubion. A chessboard coloring problem for sat solver. Soumis à *Information Processing Letters*, 2003.
- [Hao et Dorne, 1994] Jin-Kao Hao et Raphael Dorne. A new population-based method for satisfiability problems. Dans *Proc. of the 11th European Conf. on Artificial Intelligence*, pages 135–139, Amsterdam, 1994.
- [Hirsch et Kojevnikov, 2001] Edward A. Hirsch et Arist Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. PDMI preprint 9/2001, Steklov Institute of Mathematics at St.Petersburg, 2001.
- [Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [Huang et Jin, 1997] Wengi Huang et Renchao Jin. Solar, a quasi physical algorithm for sat. *Science in China (Series E)*, 2(27):179–186, 1997.
- [Jaumard *et al.*, 1994] Brigitte Jaumard, Mihnea Stan, et Jacques Desrosiers. Tabu search and a quadratic relaxation for the satisfiability problem. Dans *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 457–478, 1994.
- [Li *et al.*, 2002] Chu Min Li, Bernard Jurkowiak, et Paul W. Purdom. Integrating symmetry breaking into a dll procedure. Dans *Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT2002)*, pages 149–155, 2002.
- [Li et Anbulagan, 1997] Chu Min Li et Anbulagan. Heuristics based on unit propagation for satisfiability problems. Dans *Proc. of the IJCAI'97*, pages 366–371, 1997.
- [Li, 2000] Chu Min Li. Integrating equivalency reasoning into davis-putnam procedure. Dans *Proc. of the AAAI'00*, pages 291–296, 2000.

- [Mazure *et al.*, 1998] Bertrand Mazure, Lakhdar Saïis, et Eric Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.
- [Mitchell *et al.*, 1992] David G. Mitchell, Bart Selman, et Hector J. Levesque. Hard and easy distributions for SAT problems. Dans *Proc. of AAAI'92*, pages 459–465, 1992.
- [Selman *et al.*, 1992] Bart Selman, Hector J. Levesque, et David G. Mitchell. A new method for solving hard satisfiability problems. Dans *Proc. of the AAAI'92*, pages 440–446, San Jose, CA, 1992.
- [Selman *et al.*, 1994] Bart Selman, Henry A. Kautz, et Bram Cohen. Noise strategies for improving local search. Dans *Proc. of the AAAI, Vol. 1*, pages 337–343, 1994.
- [Selman *et al.*, 1997] Bart Selman, Henry A. Kautz, et David A. McAllester. Ten Challenges in Propositional Reasoning and Search. Dans *Proc. of the IJCAI'97*, pages 50–54, 1997.
- [Simon *et al.*, 2002] Laurent Simon, Daniel Le Berre, et Edward A. Hirsch. The sat2002 competition. Technical report, Fifth International Symposium on the Theory and Applications of Satisfiability Testing, May 2002.
- [Spears, 1996] William M. Spears. Simulated annealing for hard satisfiability problems. Dans *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 533–558, 1996.
- [Zhang, 1997] Hantao Zhang. SATO: An efficient propositional prover. Dans *Proc. of the 14th International Conference on Automated deduction*, volume 1249 of *LNAI*, pages 272–275, Berlin, 1997.