RB-SAT: Un nouveau modèle SAT basé sur les codages SAT du modèle RB

Haibo Huang, Chu Min Li, Nouredine Ould Mohamedou, Ke Xu

RB-SAT: Un nouveau modèle SAT basé sur les codages SAT du modèle RB

Haibo Huang **Chu Min Li** **Nouredine Ould Mohamedou*** **Ke Xu**

MIS, Universit de Picardie Jules Verne
5 Rue du Moulin Neuf 80000 Amiens, France.
National Lab of Software Development Environment,
Beihang University, Beijing 100083, China

chu-min.li@u-picardie.fr,wavebywind@163.com, kexu@nlsde.buaa.edu.cn

Résumé

La génération d'instances SAT difficiles est de grande importance à la fois théorique et pratique. Le Modèle RB est un modèle CSP dont les points de seuil peuvent être précisément localisés et l'on peut garantir que les instances générées à ce seuil seront difficiles. Dans ce papier, nous présentons trois différentes méthodes pour coder des instances CSP, à partir d'un Modèle RB, en instances SAT. Ensuite, nous effectuons systématiquement des comparaisons et analyses détaillées. Grâce aux résultats expérimentaux, nous avons constaté que, même en utilisant des façons différentes de coder ces instances, celles-ci demeurent assez difficiles et le pic du coût de la recherche de solutions pour ces instances correspond exactement à celui des instances CSP, ce qui signifie que les instances SAT générées de cette façon ont hérité des bonnes caractéristiques du Modèle RB. Nous avons finalement choisi la méthode la plus naturelle et « directe » de codage pour définir un modèle SAT simple appelé RB-SAT. Avec son aisance de compréhension et d'utilisation, RB-SAT est ainsi un autre modèle SAT en plus de *k*-SAT aléatoire pour étudier la difficulté du problème SAT.

1 Introduction

Le problème de la satisfiabilité (SAT), étant un des premiers problèmes NP-complets, est important aussi bien en théorie qu'en pratique. Du point de vue théorique, il joue un rôle central dans les calculs de complexités. Du point de vue pratique, il se traduit par beaucoup de nombreux problémes concrets tels que la vérification formelle, les ressources d'allocation et la gestion de l'emploi du temps qui peuvent être naturellement codés en problèmes SAT.

Depuis les années 1990, un phénomène de transition de phase, en probabilité de l'existence d'une solution, fut observé pour beaucoup de problèmes en informatique. Cette transition coïncide avec la région où la difficulté des instances s'accroît, où SAT est un des premiers problèmes susceptibles de mettre en évidence un tel phénomène.

Dans l'étude de transitions de phases pour les problèmes NP-complets, ce sont les problèmes 3-SAT aléatoires qui ont retenu le plus d'attention depuis une dizaine d'années. Beaucoup de travaux ont été réalisés pour prouver que les bornes inférieures et supérieures au voisinage des points de seuil, et les meilleures bornes inférieure et supérieure actuelles pour 3-SAT aléatoire sont respectivement de 3,52 [9] et 4,506 [2] (la valeur exacte n'étant pas encore fixée à ce jour). Dans la poursuite de la conception d'algorithmes rapides pour des problèmes NP-complets, et pour évaluer expérimentalement la performance asymptotique de ces algorithmes, il est nécessaire de générer des instances de tailles croissantes dont la difficulté peut être garantie. Principalement grâce à leur simplicité et leur difficulté, les problèmes 3-SAT aléatoires sont utilisés comme une importante source pour générer des benchmarks à soumettre aux différents algorithmes ou solveurs (ils représentent une des trois principales catégories de benchmarks adoptées par la *SAT Competition*[1]).

En fait, 3-SAT aléatoire a grandement motivé le développement d'algorithmes tels que satz [11] et adaptg2wsat

*nouredine.ould@u-picardie.fr

[1] http ://www.satcompetition.org

[12] au cours des dernières années. Toutefois, l'absence de structure dans les instances 3-SAT aléatoires fait qu'elles sont tout à fait différentes de celles générées par un codage SAT de problèmes du monde réel, de sorte qu'un algorithme efficace pour 3-SAT aléatoire ne l'est pas nécessairement pour les problèmes SAT issus du monde réel. Pour fournir une bonne alternative aux critères purement aléatoires, le problème de quasigroup (*latine Squares* ou QCP) (voir par exemple [10]) a été présenté comme un domaine de benchmarks structurés [1, 6], ce qui a grandement fait progresser l'étude du problème SAT, surtout empiriquement. Cependant, par rapport au 3-SAT aléatoire, ce domaine n'est pas encore assez simple à comprendre et à utiliser, et le codage SAT de ce problème n'est pas encore assez affiné pour l'étude asymptotique du comportement d'un algorithme. Par exemple, le codage d'une instance SAT $\mathcal{F}_1$ du *latine Squares* d'ordre 20 utilise naturellement 8000 (20^3) variables booléennes, alors qu'un codage d'une instance SAT $\mathcal{F}_2$ du *latine Squares* d'ordre 21 fait naturel usage de 9261 (21^3) variables booléennes. Ceci est différent de 3-SAT aléatoire pour lequel on peut continuellement augmenter la taille de l'instance par l'ajout de 50 variables pour étudier asymptotiquement le comportement d'un algorithme.

Il est donc utile de fournir un effort de plus afin de trouver un nouveau modèle SAT avec de meilleures fonctionnalités pour permettre de mieux étudier le problème SAT, tout en étant presque aussi simple que le modèle 3-SAT aléatoire. Idéalement, un algorithme efficace pour ce genre de benchmarks SAT devrait être également efficace pour une large palette de problèmes SAT aléatoires et structurés.

Le Problème de Satisfaction de Contraintes[2] (CSP) est une généralisation du problème SAT. Le CSP peut être transformé en un problème SAT en utilisant des codages différents, tels que le codage direct, le codage de support et le codage logarithmique. Dans [20], un modèle de CSP appelé Modèle RB a été proposé. Contrairement au 3-SAT aléatoire, le Modèle RB a des points de seuil qui peuvent être précisément situés. En outre, à l'heure actuelle de nombreuses études sur des algorithmes incomplets s'appuient beaucoup sur des expérimentations et analyses empiriques pour des instances satisfiables mais assez difficiles. Des travaux antérieurs [19] montrent que le Modèle RB peut également être utilisé pour générer des instances difficiles et satisfiables en utilisant une stratégie très simple. Puisque le Modèle RB a les avantages mentionnés ci-dessus, il serait donc pertinent d'explorer si ces propriétés intéressantes se retrouvent lorsque l'on traduit en instances SAT des instances générées par les modèles RB (les différents codages pourraient en effet avoir des propriétés très différentes). Dans ce papier, nous effectuons une étude plus détaillée et systématique utilisant trois méthodes de codage, à la recherche des différents paramètres de contrôle, de la transition de phase en satisfiabilité, du rapport de backbone ainsi que de la difficulté au voisinage du seuil. Selon nos connaissances, c'est la première fois qu'une étude est réalisée par rapport aux comportements respectifs de ces trois méthodes de codage (direct, de support et logarithmique) en utilisant des instances asymptotiquement dures. En outre, nous proposons dans ce papier un modèle simple et aléatoire issu des fonctionnalités du Modèle RB, et nous espérons que ce nouveau modèle sera utilisable par la suite pour l'étude des problèmes SAT.

Le reste de ce papier est organisé comme suit. Tout d'abord dans la section 2, nous introduirons quelques définitions de base ainsi que le Modèle RB. Ensuite, un aperçu des codages CSP en SAT est fait dans la section 3. Par la suite, nous présenterons nos résultats expérimentaux ainsi que l'analyse dans la section 4. Après cela, nous proposons un nouveau modèle SAT aléatoire appelé RB-SAT, et précisons ses avantages par rapport au 3-SAT aléatoire à la section 5. Enfin, dans la section 6 nous ferons nos conclusions et discuterons des travaux futurs.

2 Préliminaires

2.1 SAT et k-SAT aléatoire

Une instance SAT est une formule propositionnelle sous Forme Normale Conjonctive CNF (en anglais : Conjonctive Normal Form). Le problème SAT consiste à déterminer si une telle formule est satisfiable ou non. En termes plus formels, cela se résume dans les définitions suivantes :

Variables booléennes : Variables pouvant avoir pour valeur l'une des valeurs de vérité *vrai* ou *faux*.

Littéraux : Variables (littéraux positifs, par exemple P) ou leurs négations (littéraux négatifs, par exemple $\neg P$).

Clauses : Disjonction de littéraux, par exemple $P \vee \neg Q$.

Formule CNF : Conjonction de clauses, par exemple $(P \vee \neg Q) \wedge (P \vee Q)$.

Problème SAT : Étant donnée une formule CNF F, demander s'il existe une affectation de valeurs de vérité à des variables booléennes telles que toutes les clauses de F soient satisfaites (évaluées à *vrai*). Si tel est le cas, la formule F est dite satisfiable, sinon elle est dite insatisfiable.

[2] Constraint Satisfaction Problem

k-**SAT** : C'est le problème SAT restreint à des clauses avec exactement *k* littéraux (des clauses ayant toutes une longueur exactement égale à *k*). Il est assez connu que le problème *k*-SAT est un problème NP-complet pour $k \geq 3$. Pour beaucoup de problèmes NP-complets, la probabilité de générer aléatoirement des instances satisfiables montre une forte transition de phase quand un paramètre de contrôle est varié, à partir d'une région où la probabilité est près de 1 à une région où la probabilité est près de 0. La transition de phase correspond en général à un pic de coût de recherche pour les différents algorithmes les plus connus de la littérature, c'est-à-dire que les instances générées grâce aux paramètres de contrôle près de la valeur critique sont les plus difficiles à résoudre.

Le problème *k*-SAT aléatoire est une distribution probabiliste des instances *k*-SAT. Une instance *k*-SAT aléatoire avec *n* variables et *m* clauses est construite par la sélection de *m* clauses de manière uniforme et indépendante de l'ensemble de toutes les *k*-clauses possibles sur *n* variables, où chaque clause a exactement *k* littéraux sans redondance (sans répétition d'occurrence) de variables dans une même clause. Pour *k*-SAT aléatoire, le ratio des clauses par rapport aux variables (c'est-à-dire m/n) agit souvent comme le paramètre de contrôle.

2.2 Le problème CSP

Un Problème de Satisfaction de Contraintes (CSP) est généralement défini comme un triplet $\langle X, D, C \rangle$, où $X=\{x_1, ..., x_n\}$ est un ensemble de variables, D est un domaine de valeurs et $C=\{C_1, ..., C_p\}$ est un ensemble de contraintes. Chaque contrainte $C_i=\langle S_i, R_i \rangle$ concerne un ensemble de variables $S_i=\{x_{i_1}, ..., x_{i_k}\}$ (où k est l'arité de C_i) et a une relation associée R_i qui spécifie les tuplets de valeurs compatibles pour ces variables dans S_i.

Une contrainte est dite satisfaite si le tuplet des valeurs assigné aux variables dans cette contrainte est compatible. Une solution au CSP est une affectation de valeurs de vérité à toutes les variables telle que toute contrainte soit satisfaite. Un CSP ayant au moins une solution est dit satisfiable (sinon, insatisfiable). Le but du CSP est de trouver une solution, s'il est satisfiable, ou de prouver qu'il est insatisfiable. En général, le problème CSP est NP-complet.

2.3 Le Modèle RB

Modèle RB [20] : Une classe d'instances CSP aléatoires, générées suivant le Modèle RB qui s'exprime par $RB(k, n, \alpha, r, p)$, où pour toute instance :

- $k \geq 2$ est l'arité de chaque contrainte,
- $n \geq 2$ est le nombre de variables,
- $\alpha > 0$ détermine la taille du domaine $d=n^{\alpha}$ de chaque variable,
- $r > 0$ détermine le nombre $m=r.n.ln(n)$ de contraintes,
- $0 < p < 1$ détermine le nombre $t=pd^k$ de tuplets rejetés (incompatibles ou exclus) dans chaque relation.

On suppose que tous les domaines des variables contiennent le même nombre de valeurs $d=n^{\alpha}$ dans le Modèle RB. Dans ce papier, nous limitons notre attention au cas binaire du Modèle RB, c'est-à-dire le cas où k=2. La génération aléatoire des instances CSP dans le Modèle RB est effectuée selon les deux étapes suivantes :

Étape 1. Sélectionner avec répétition $m=rnln(n)$ contraintes aléatoires. Chaque contrainte aléatoire est formée par sélection sans répétition de k variables parmi les n variables du problème.

Étape 2. Pour chaque contrainte, sélectionner uniformément et sans répétition $t=pd^k$ tuplets incompatibles de valeurs.

L'instance peut être forcée à être satisfiable par l'affectation aléatoire d'une valeur à chaque variable, ensuite garder chaque contrainte satisfaite lors de la sélection des tuplets incompatibles de valeurs pour cette contrainte dans l'*Étape 2*.

L'existence de transitions de phase dans le Modèle RB fut prouvée dans [20] et les points de seuil exacts y sont donnés. Plus précisément, nous avons les théorèmes suivants : (où Pr(Sat) exprime la probabilité qu'une instance aléatoire $P \in RB(k, n, \alpha, r, p)$ soit satisfiable) :

Theorem 2.1 *Si k,* $\alpha > \frac{1}{k}$ *et* $p \leq \frac{k-1}{k}$ *sont des constantes alors*

$$\lim_{n \to \infty} \Pr(Sat) = \begin{cases} 1 & if\ r < r_{cr} \\ 0 & if\ r > r_{cr} \end{cases}$$

où $r_{cr} = -\frac{\alpha}{\ln(1-p)}$.

Theorem 2.2 *Si k,* $\alpha > \frac{1}{k}$ *et* $p_{cr} \leq \frac{k-1}{k}$ *sont des constantes alors*

$$\lim_{n \to \infty} \Pr(Sat) = \begin{cases} 1 & if\ p < p_{cr} \\ 0 & if\ p > p_{cr} \end{cases}$$

où $p_{cr} = 1 - e^{-\frac{\alpha}{r}}$.

3 Codages de CSP en SAT

3.1 Codage direct

Dans [18] Walsh avait introduit le codage direct qui est considéré comme étant, à la fois, le codage le plus naturel et le plus largement utilisé. Plus précisément, une variable SAT x_{v_i} est vraie si et seulement si la valeur de vérité i est affectée à la variable CSP v (i et j étant

dans $dom(v)$=$\{0,...,d-1\}$). Le codage direct est constitué de trois types de clauses. Les « au-moins-une » clauses $x_{v_0} \vee x_{v_2} \vee ... \vee x_{v_{d-1}}$ qui expriment le fait que chaque variable CSP doit prendre au moins une valeur du domaine, les « au-plus-une » clauses $\overline{x}_{v_i} \vee \overline{x}_{v_j}$ signifiant que toute variable CSP peut prendre au plus une valeur du domaine et les clauses conflictuelles (ou de « conflit ») $\overline{x}_{v_i} \vee \overline{x}_{w_j}$ qui sont, quand à elles, utilisées afin d'exprimer les conflits.

Par exemple, on considère le problème CSP suivant : $A > B$, avec A et B appartenant à l'ensemble de valeurs $\{1,2,3\}$. Le codage direct est exprimé comme suit :
au-moins-une : $a_1 \vee a_2 \vee a_3, b_1 \vee b_2 \vee b_3$
au-plus-une : $\neg a_1 \vee \neg a_2, \neg a_1 \vee \neg a_3, \neg a_2 \vee \neg a_3, \neg b_1 \vee \neg b_2, \neg b_1 \vee \neg b_3, \neg b_2 \vee \neg b_3$
conflit : $\neg a_1 \vee \neg b_1, \neg a_1 \vee \neg b_2, \neg a_1 \vee \neg b_3, \neg a_2 \vee \neg b_2, \neg a_2 \vee \neg b_3, \neg a_3 \vee \neg b_3$

3.2 Codage de support

Similaire au codage direct, le codage de support génère les clauses « au-moins-une » et « au-plus-une » comme le codage direct. En revanche, ce codage remplace les clauses de « conflit » du codage précédent par les clauses de support définies comme suit. Si $i_1 ... i_k$ sont des valeurs de support dans le domaine de la variable CSP v pour la valeur j dans le domaine de la variable CSP w, alors ajouter la clause de support (ou de soutien) $x_{v_{i_1}} \vee ... \vee x_{v_{i_k}} \vee \overline{x}_{v_{w_j}}$.

Il est donc évident que le codage direct et celui du support semblent avoir beaucoup de similitudes. En effet, il fut montré par Gent [5] que les clauses de support peuvent être dérivées à partir du codage direct. Ce qui suit montre le codage du support correspondant à l'exemple précédent.
au-moins-une : $a_1 \vee a_2 \vee a_3, b_1 \vee b_2 \vee b_3$
au-plus-une : $\neg a_1 \vee \neg a_2, \neg a_1 \vee \neg a_3, \neg a_2 \vee \neg a_3, \neg b_1 \vee \neg b_2, \neg b_1 \vee \neg b_3, \neg b_2 \vee \neg b_3$
support : $\neg a_1, \neg a_2 \vee b_1, \neg a_3 \vee b_1 \vee b_2, \neg b_1 \vee a_2 \vee a_3, \neg b_2 \vee a_3, \neg b_3$

3.3 Le codage logarithmique

Dans le codage logarithmique, nous utilisons m=$\lceil \log_2 d \rceil$ variables logiques pour représenter les domaines et chacune des 2^m combinaisons représente une affectation possible. Le codage logarithmique utilise un nombre logarithmique au lieu d'un nombre linéaire de variables booléennes pour coder les domaines. Ce qui permet de générer des modèles de taille plus petite. Pour chaque valeur de bit CSP variable/domaine, il existe une variable SAT correspondante, et chaque conflit a une clause SAT qui lui correspond également. Les clauses « au-moins-une » et « au-plus-une » sont inutiles pour le codage logarithmique. Au lieu de cela, ce codage dispose des clauses dites à « valeur interdite » [15] qui sont nécessaires pour exclure les valeurs en trop quand la cardinalité des domaines n'est pas une puissance de deux. Dans l'exemple utilisé, le codage logarithmique est montré comme suit :
valeur-interdite : $a_1 \vee a_0, b_1 \vee b_0$
conflit : $a_1 \vee \neg a_0 \vee b_1 \vee \neg b_0, a_1 \vee \neg a_0 \vee \neg b_1 \vee b_0, a_1 \vee \neg a_0 \vee \neg b_1 \vee \neg b_0, \neg a_1 \vee a_0 \vee \neg b_1 \vee b_0, \neg a_1 \vee a_0 \vee \neg b_1 \vee \neg b_0, \neg a_1 \vee \neg a_0 \vee \neg b_1 \vee \neg b_0$

4 Expérimentations

Il est bien connu que les codages SAT d'un problème peuvent avoir des propriétés très différentes : certains facteurs de la difficulté peuvent être supprimés, mais certaines difficultés de résolution peuvent être introduites ou ajoutées. Par exemple, lorsque le problème de parité DIMACS, intrinsèquement facile, a été codée en un problème SAT *challenge* pour les solveurs SAT [16], il fut observé dans [1] que le problème des quasigroups devient plus facile à résoudre par satz après l'avoir codé en SAT. Les caractéristiques des codages qui peuvent être efficacement résolus représente aussi un *challenge* donné dans [16].

Dans cette section, nous donnons quelques résultats expérimentaux représentatifs des différents codages. Quelques analyses théoriques sont déjà faites dans [21], utilisant le codage direct et concernant les instances CSP et des résultats empiriques sont présentés dans [19]. Afin de réaliser une meilleure comparaison et d'élargir le spectre de l'applicabilité des résultats précédents, nous réalisons des expérimentations comparatives et modifions la plupart des paramètres de contrôle en plus de ceux du [19].

Ces expérimentations ont été réalisées sur un *PC Pentium IV 3 GHz 1GMb* sous *Windows XP Professional SP2*. Tous les points de données dans les figures, de 1 à 8, sont obtenus en utilisant 50 instances. Les algorithmes (solveurs) utilisés dans ces expérimentations sont bien connus dans la littérature, incluant à la fois des algorithmes complets (satz [11] et zchaff [14]) et d'autres incomplets (walksat [17] et adaptnovelty [8]).

4.1 Coût de recherche des codages SAT pour le Modèle RB

Il a été montré dans [19] que les plus difficiles instances CSP sont situées assez près du seuil théorique, et que les instances forcées et les instances non forcées ont tendance à y avoir des difficultés similaires. Afin de voir si ces résultats sont toujours valables pour les instances SAT obtenues en utilisant les différents codages, nous réalisons des expérimentations comparatives en utilisant les trois codages présentés ci-dessus (direct, de support et logarithmique).

Dans les figures 1 et 2 (gauche), nous avons étudié la difficulté de résoudre avec zchaff les instances SAT du Modèle RB générées au voisinage du seuil théorique $p_{cr} \approx 0.23$ donné par le théorème 2.2 pour k=2, α=0.8,

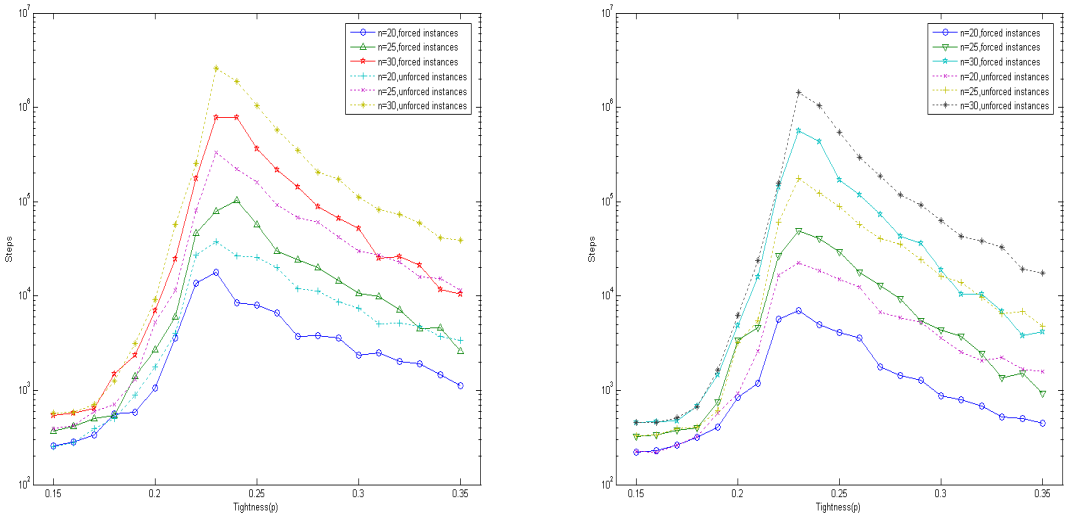

FIG. 1 – Moyenne du coût de recherche d'une solution pour des instances codées en $RB(2,\{20,25,30\},0.8,3,p)$ avec zchaff, utilisant le codage direct (à gauche) et le codage de support (à droite)

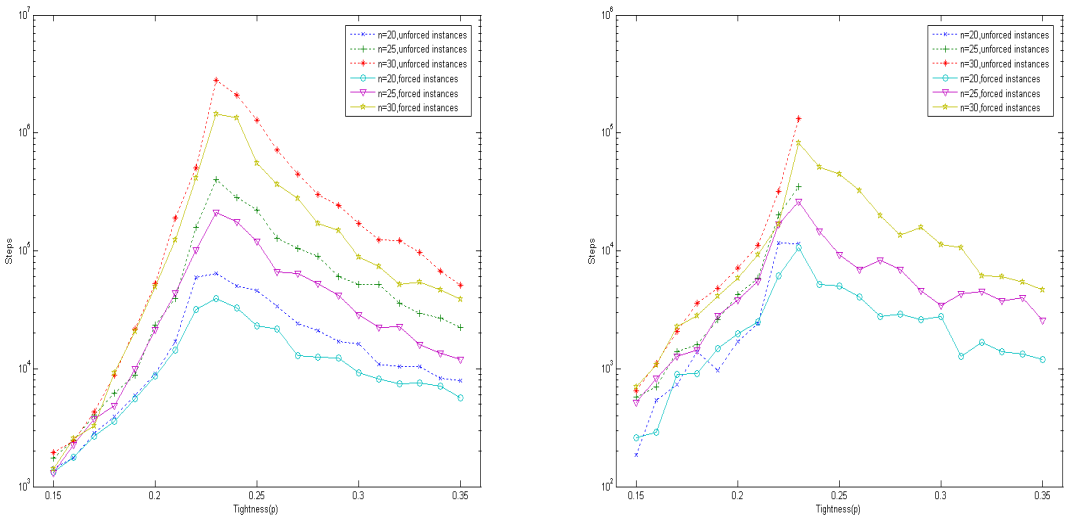

FIG. 2 – Moyenne du coût de recherche d'une solution pour des instances codées en $RB(2,\{20,25,30\},0.8,3,p)$ avec zchaff, utilisant le codage logarithmique (à gauche) et adaptnovelty utilisant le codage direct (à droite)

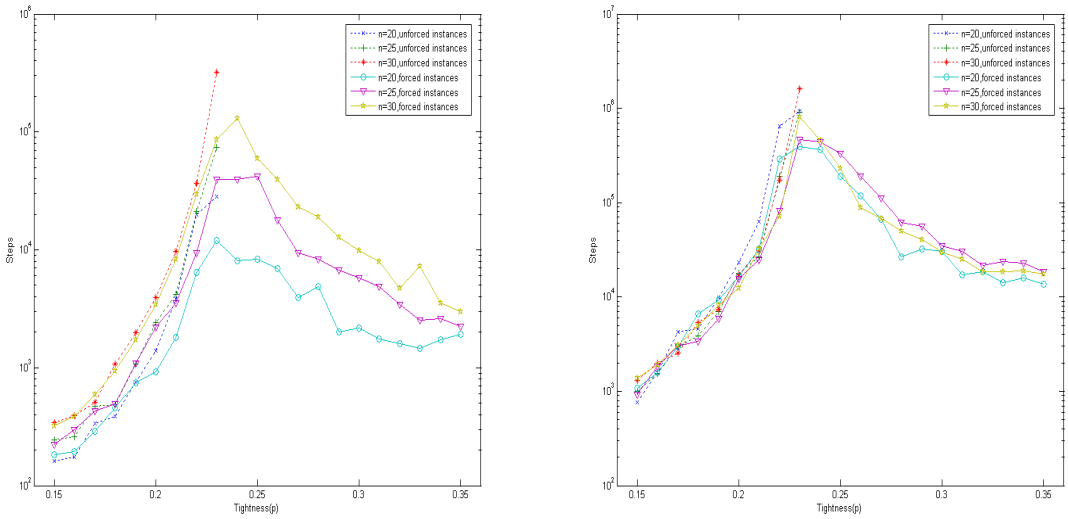

FIG. 3 – Moyenne du coût de recherche d'une solution pour des instances codées en $RB(2,\{20,25,30\},0.8,3,p)$ avec adaptnovelty, utilisant le codage du support (à gauche) et le codage logarithmique (à droite)

r=3 et $n \in \{20,30,40\}$.

Ces figures montrent clairement que quelque soit la méthode de codage utilisée, la plupart des instances les plus difficiles apparaissent toujours près du point du seuil théorique et il y a peu de différence, en difficulté, entre les instances forcées et celles qui ne le sont pas. Une telle difficulté augmente de façon exponentielle selon n (utilisation d'une échelle logarithmique). Dans les figures 2 (droite) et 3, nous utilisons le solveur incomplet adaptnovelty à la place de zchaff pour résoudre les instances, et il peut être constaté que les résultats sont sensiblement identiques. Pour le codage direct (que nous utiliserons pour définir le nouveau modèle SAT plus loin dans la section 5), nous avons plus d'expérimentations avec les deux algorithmes bien connus tels que satz et walksat, et les résultats sont présentés dans la figure 4.

Nous avons également effectué des expérimentations pour tester la satisfiabilité des instances non forcées. Leurs résultats sont montrés dans la figure 5. Comme attendu, au seuil théorique $p_{cr} \approx 0.23$ pour $RB(2,\{20,25,30\},0.8,3,p)$, nous observons la plus nette (aigüe) transition de phase en satisfiabilité. Il peut être constaté que la largeur de la région contenant la transition de phase se rétrécit (diminue) lorsque n est assez grand, ce qui est très similaire à ce qui a été trouvé pour le k-SAT aléatoire.

4.2 Transition de phase dans le backbone des instances forcées

Pour les instances forcées, nous étudions le rapport backbone dans nos expérimentations. La notion du backbone d'un problème SAT fut introduite dans [13] pour désigner le rapport des variables qui prennent les mêmes valeurs dans toutes les solutions. Le rapport backbone (ratio des variables du backbone au nombre total de variables) est une propriété des problèmes CSP et SAT qui est bien définie pour les distributions satisfiables. Le phénomène de transition de phase dans le rapport du backbone a été observé dans [1].

Dans nos expérimentations, nous avons constaté la présence de ce phénomène quelque soit la méthode de codage adoptée.
Les figures 6 et 7 (gauche) montrent le rapport du backbone comme une fonction d'étroitesse (d'exiguïté) p des différents nombres de variables n. Une forte transition de phase apparaît dans le rapport du backbone, qui correspond au pic du coût de difficulté au cours de la recherche montrée ci-dessus (où p=0.23). On peut également observer que plus n croît plus la largeur de la région de transition de phase diminue.

De plus, nous avons trouvé que les sous figures (gauche et doite) de la figure 6 sont presque identiques, et il semble que le codage direct et le codage de support ont effectivement des points fondementaux en commun. Cet intriguant phénomène mérite une recherche plus approfondie.

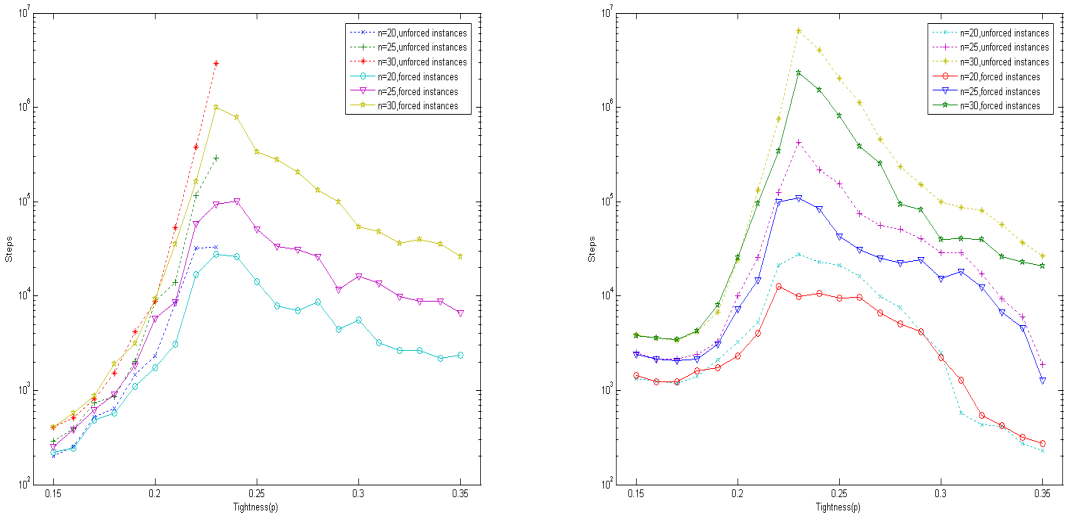

FIG. 4 – Moyenne du coût de recherche d'une solution pour des instances codées en $RB(2,\{20,25,30\},0.8,3,p)$ avec walksat (à gauche) et satz (à droite), utilisant le codage direct

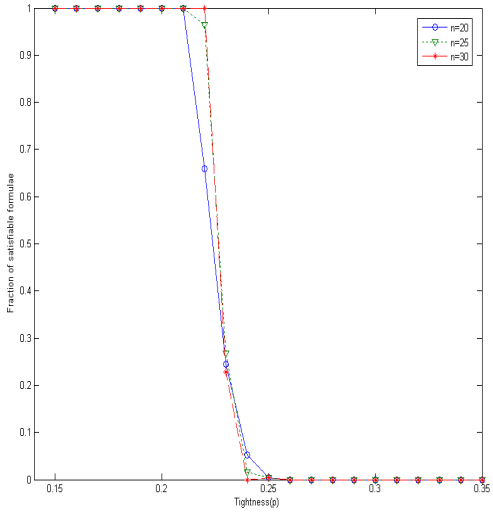

FIG. 5 – Transition de phase en satisfiabilité pour $RB(2,\{20,25,30\},0.8,3,p)$

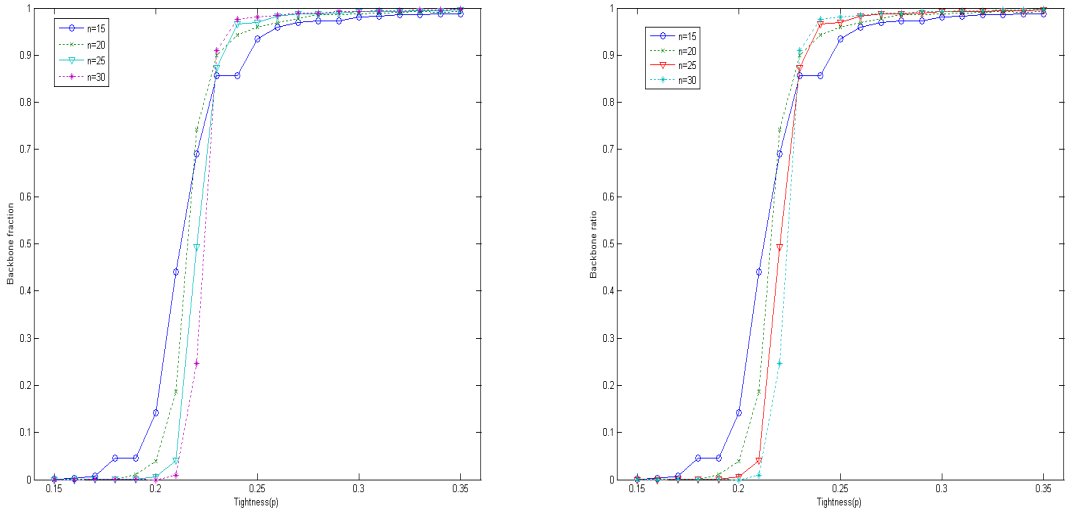

FIG. 6 – Transition de phase dans le rapport du backbone pour les instances codées en $RB(2,\{15,20,25,30\},0.8,3,p)$, utilisant le codage direct (à gauche) et le codage de support (à droite)

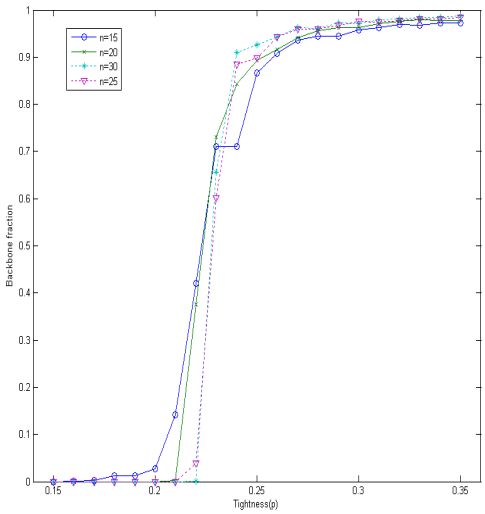

FIG. 7 – Transition de phase dans le rapport du backbone pour les instances codées en $RB(2,\{15,20,25,30\},0.8,3,p)$, utilisant le codage algorithmique

4.3 Difficulté des instances à proximité du seuil

Les figures 1 à 5 ont montré que la difficulté des instances forcées est tout à fait similaire à celle des instances non forcées, laquelle s'accentue exponentiellement lorsque n augmente. Pour confirmer et illustrer l'étendu du spectre de l'applicabilité de ce résultat, nous nous sommes concentrés sur un point juste en dessous du seuil. Pour cela, différentes valeurs de paramètres α et r ont été choisies. En outre, nous voulons aussi étudier au préalable la façon dont les coûts de recherche diffèrent lorsque les différents codages sont utilisés.

La figure 8 présente le coût de recherche de solution avec zchaff, à la fois, pour les instances SAT forcées et les non forcées générées grâce au Modèle RB avec $p_{cr} - 0.01 \approx 0.40$ pour k=2, α=0.8, r=1.5 et $n \in [20...40]$, utilisant les trois différents codages. Une fois de plus, le résultat confirme ce qui a été observé auparavant ; à savoir que les deux types d'instances (forcées et non forcées) ont une complexité exponentielle très similaire à proximité du seuil.

La différence entrainée par les codages montre que, généralement, le codage logarithmique produit des instances difficiles, tandis que le codage de support en génère des plus faciles. Ceci concorde bien avec les comparaisons expérimentales faites dans [4, 5, 7] (ces comparaisons avaient été réalisées par des chercheurs différents, utilisant des méthodologies tout à fait différentes). Avec un examen plus attentif, on peut voir que, bien que ces lignes commencent à partir de différents points, elles sont presque parallèles lorsque n augmente suffisamment.

On peut donc conclure que la différence en performance entre les codages reste relativement identique, même quand la taille du problème grandit (ce qui génère des instances difficiles). Cela concorde bien avec ce qui a été trouvé auparavant, et d'après nos connaissances, c'est la première fois que le comportement asymptotique des trois codages est étudié à l'aide des instances dures.

5 Un nouveau modèle SAT aléatoire

Les avantages du Modèle RB résident dans sa simplicité, la difficulté des instances qu'il produit et ses bonnes propriétés théoriques. Les résultats expérimentaux de la section 4 montrent que quelque soit la méthode de codage utilisée, les instances SAT obtenues grâce au Modèle RB héritent de bonnes caractéristiques de RB : aisance d'utilisation, précision en transition de phase et difficulté de résolution. Nous pouvons alors définir un modèle SAT simple correspondant à chacune des trois méthodes de codage. Le modèle suivant correspond au codage direct du Modèle RB (avec k=2) :

Étape 1. *Générer n ensembles disjoints de variables booléennes dont chacun a pour cardinalité* n^{α} *(où* $\alpha > 0$ *est une constante). Pour tout ensemble, générer une clause qui est la disjonction de toutes les variables de cet ensemble et pour toutes deux variables x et y dans ce même ensemble, générer une clause binaire* $\neg x \vee \neg y$.

Étape 2. *Sélectionner aléatoirement deux ensembles disjoints et générer sans répétitions* $pn^{2\alpha}$ *clauses de la forme* $\neg x \vee \neg z$ *où x et z sont deux variables sélectionnées aléatoirement et respectivement à partir des deux ensembles (où* $0 < p < 1$ *est une constante) ;*

Étape 3. *Faire l'Étape 2 (avec répétitions)* $rn\ln n$-1 *fois (où* $r > 0$ *est une constante).* Ce modèle, que nous nommons RB-SAT(n, α, r, p) (puisqu'il est basé sur le Modèle RB), est évidemment très facile à comprendre, même sans aucune connaissance du CSP ou de l'encodage, et peut donc être utilisé très convenablement dans les études théoriques et expérimentales du problème SAT. Nous pensons que RB-SAT fournit un autre bon modèle SAT aléatoire en plus du k-SAT aléatoire. À l'instar du k-SAT, il a été prouvé dans [21] que le codage SAT direct des formules de type RB n'ont presque certainement pas d'arbres de résolution de preuves en-dessous de l'exponentielle, ce qui implique que RB-SAT est difficile pour les résolutions d'arbres. Les résultats expérimentaux de la section 4 confirment que les instances RB-SAT sont difficiles. Étant donné que le codage de CSP en SAT n'a pas d'impact sur la satisfiabilité, nous savons que les transitions de phase exactes montrées dans [20] existent également dans RB-SAT, c'est à dire que les théorèmes 2.1 et 2.2 s'appliquent facilement à RB-SAT(n, α, r, p) pour donner la valeur exacte du seuil correspondant à la transition de phase de satisfiabilité, ce qui est très utile pour l'étude de la difficulté des problèmes SAT. On note que la valeur exacte du seuil correspondant à la transition de phase de la satisfiabilité de k-SAT demeure inconnue à ce jour.

Les instances de RB-SAT(n, α, r, p) contiennent $n^{1+\alpha}$ variables et $O(n^{1+2\alpha}\ln(n))$ (c'est à dire, $n(n^{\alpha}$-1)/2+1)+$rn\ln(n)pn^{2\alpha}$) clauses. En pratique, nous utilisons $0.5 < \alpha < 1$ pour générer des instances difficiles au seuil selon les théorèmes 2.1 et 2.2. Ainsi, le nombre de variables booléennes augmente relativement lentement lorsque n augmente.

En outre, les instances RB-SAT peuvent être facilement générées en utilisant la même stratégie que RB. Ces instances sont assurément satisfiables et ayant une difficulté similaire à celle des instances non forcées. En plus, un phénomène de transition de phase dans le rapport de backbone de ces instances est observé, caractérisant ainsi leur difficulté. Les pics de difficulté des instances forcées

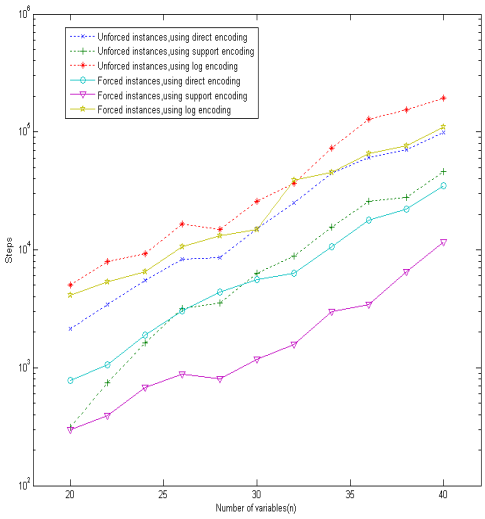

FIG. 8 – Coût moyen de recherche de solution pour des instances codées en $RB(2,\{20...40\},0.8,1.5,p_{cr}-0.01)$ avec zchaff

et celui des instances non forcées coïncident.

La figure 9 (gauche) compare 3-SAT aléatoire avec un ratio clause/variable de 4.25 et RB-SAT($n,0.8,3,0.23$) utilisant trois solveurs de l'état-de-l'art : minisat [3], satz et adaptg2wsat+ [12]. Notons que les instances 3-SAT aléatoires sont au seuil empirique lorsque les instances RB-SAT sont au seuil donné dans le théorème 2.2. Les instances comparées de 3-SAT et RB-SAT ont le même nombre de variables, correspondant à n=20, 23, 25, 28, 30, 32, 35, 38, et 40 dans RB-SAT. La figure 9 (droite) compare les instances non forcées mais satisfiables de RB-SAT et les instances forcées de RB-SAT avec les même paramètres, après utilisation de satz ou minisat pour filtrer les instances insatisfiables non forcées de RB-SAT.

Minisat est choisi pour sa haute performance dans les récentes *SAT competition*. Nous en utilisons la récente version 2.8. Quant au solveur satz, il est choisi pour sa haute performance à la fois pour les instances 3-SAT aléatoires et pour les problèmes structurés tels que celui des quasi-groups. Le solveur adaptg2wsat+ est choisi puisqu'il résout le plus grand nombre d'instances entre tous les solveurs basés sur les algorithmes de la Recherche Locale dans *SAT competition* de 2007 [3]. À chaque point, où 250 instances 3-SAT aléatoires et 250 instances RB-SAT sont résolues, la moyenne des temps d'exécution est affichée.

Sur la figure 9, nous observons que :

– Comme pour 3-SAT aléatoire, on peut sans cesse augmenter la taille des instances RB-SAT tout en les maintenant solubles. La palette de taille des instances RB-SAT solubles est encore plus grande que celle du 3-SAT aléatoire, car les instances RB-SAT difficiles de 750 variables restent solubles, tandis que les instances 3-SAT aléatoires difficiles peuvent difficilement dépasser 500 variables. En outre, il existe de nombreux exemples de la même taille, ce qui est très utile pour évaluer le comportement asymptotique d'un algorithme.
– Satz est plus rapide que minisat pour 3-SAT aléatoire, alors que minisat est plus rapide que satz pour RB-SAT. Ceci est dû à l'absence de structure dans 3-SAT aléatoire, tandis qu'une instance RB-SAT peut avoir quelques structures, par exemple pour chaque ensemble disjoint de variables booléennes, il y a des clauses imposant qu'une et une seule variable dans cet ensemble peut prendre la valeur 1. En d'autres termes, afin de résoudre efficacement les instances RB-SAT, un solveur devrait faire mieux que minisat pour la partie aléatoire de RB-SAT et être plus performant que satz pour traiter les dépendances des variables dans RB-SAT, motivant ainsi le développement de solveurs efficaces à la fois pour les problèmes structurés et les problèmes aléatoires.
– Pour le solveur adaptg2wsat+, les instances RB-SAT sont plus difficiles que les 3-SAT aléatoires, ce qui signifie que la recherche locale devrait être améliorée afin de traiter de manière plus efficace les dépendances des variables. En effet, ceci est un des dix *challenges* donnés dans [16]. RB-SAT peut donc être un bon moyen pour répondre à ce défi.

[3] www.satcompetition.org

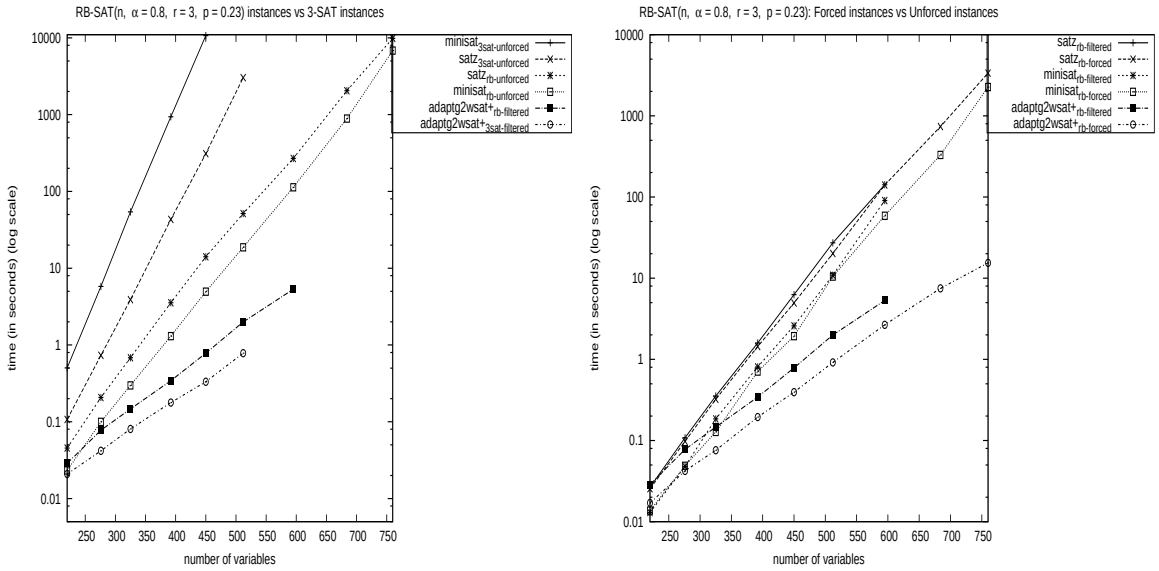

FIG. 9 – Comparaison de 3-SAT aléatoire avec RB-SAT (à gauche), et des instances non forcées RB-SAT avec des instances forcées RB-SAT (à droite), en terme de moyenne des temps d'exécution en seconde des 250 instances résolues par solveur à chaque point. Ces expérimentation ont été réalisées sur un mac pro 2.8 Ghz avec 2x4 processeurs.

En effet, puisqu'il n'est pas très facile d'augmenter le nombre de variables pour de nombreux autres problèmes SAT structurés tout en gardant ces problèmes solubles, on peut assez facilement générer beaucoup d'instances difficiles et satisfiables de RB-SAT tout en augmentant soigneusement leurs tailles (en les maintenant toujours satisfiables) pour mieux évaluer le comportement asymptotique d'une approche de recherche locale, lors du traitement des dépendances des variables. De plus, ces instances sont également aléatoires, permettant ainsi à l'approche développée de sauvegarder son efficacité pour les instances aléatoires. Observons que même si les instances RB-SAT forcées sont plus faciles que les non forcées satisfiables pour adaptg2wsat+, la différence en performance de adaptg2wsat+ semble être la même quand les instances deviennent assez grandes.

– RB-SAT est plus similaire aux codages des formules SAT relevant des problèmes issus du monde réel que 3-SAT aléatoire. Ainsi, un algorithme efficace pour RB-SAT devrait être probablement plus efficace pour les problèmes SAT du monde réel qu'un algorithme efficace uniquement pour 3-SAT aléatoire.

Dans la pratique, on peut modifier légèrement l'étape 3 pour choisir les deux différents ensembles disjoints sans répétition, les instances RB-SAT générées deviennent alors nettement plus difficiles, ce qui devrait être encore plus utile pour une meilleure mise au point des solveurs SAT. On admet que les théorèmes 2.1 et 2.2 sont toujours valables avec une telle modification car, pour des n très grands, il y a très peu de différence avec ou sans répétition.

6 Conclusions et travaux futurs

Il est bien connu que les codages SAT pour un problème donné peuvent avoir différentes propriétés de calcul. Dans ce papier, basé sur le Modèle RB, nous faisons des analyses systématiques et détaillées de trois méthodes de codage de CSP en SAT. Il est démontré que quelque soit la méthode de codage utilisée, les instances SAT obtenues partagent les bonnes caractéristiques du Modèle RB, y compris la croissance exponentielle de la difficulté des instances qui atteint son summum à la pointe du seuil. De plus, pour les instances forcées, nous avons étudié le rapport de backbone et avons observé la transition de phase. Nous avons également comparé les comportements asymptotiques liés aux trois méthodes de codage et avons trouvé que la différence en performance de ces codages demeure relativement la même quand la taille du problème devient grande. En nous basant sur le codage direct, nous avons proposé un codage simple et naturel que nous avons appelé RB-SAT. Nous en avons ainsi présenté les avantages par rapport au 3-SAT aléatoire. Nos travaux futurs pourront inclure les analyses théoriques de RB-SAT, par exemple, la preuve de la transition de phase dans le rapport du backbone, l'analyse des moyennes du temps de la complexité pour les différents algorithmes (ou heuristiques), preuve des théorèmes 2.1 et 2.2 quand l'*Étape 1* est modifiée pour sélectionner deux ensembles disjoints de variables sans répétition et fournir (de nouvelles) ou améliorer les bornes inférieures de complexité déjà existantes. Nous utiliserons également le Modèle RB-SAT pour développer de nouvelles approches pour la résolution des problèmes SAT.

Note

Cette recherche fut particulièrement supportée par le *National 973 Program of China (Grant No. 200532CB1902)*. Une partie de ce travail a été réalisée en collaboration au laboratoire MIS à l'Université de Picardie Jules Verne.

Références

[1] D. Achlioptas, C. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 256–301. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999, 2000.

[2] O. Dubois and J. Mandler. Typical random 3-SAT formulae and the satisfiability threshold. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 126–127. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2000.

[3] N. Een and N. Sorensson. An extensible SAT-solver. *Proceedings of SAT-03*, pages 502–508, 2003.

[4] M.D. Ernst, T.D. Millstein, and D.S. Weld. Automatic SAT-compilation of planning problems. In *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1169–1176, 1997.

[5] I.P. Gent. Arc consistency in SAT. In *Ecai 2002 : 15th European Conference on Artificial Intelligence, July 21-26, 2002, Lyon France : Including Prestigious Applications of Intelligent Systems (PAIS 2002) : Proceedings*. IOS Press, 2002.

[6] C.P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 221–227, 1997.

[7] H.H. Hoos. SAT-encodings, search space structure, and local search performance. In *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 296–303. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.

[8] H.H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proceedings of the national conference on artificial intelligence*, pages 655–660. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999, 2002.

[9] A.C. Kaporis, L.M. Kirousis, and E.G. Lalas. The probabilistic analysis of a greedy satisfiability algorithm. *Random Structures and Algorithms*, (4) :444–480, 2006.

[10] SR Kumar. Approximating latin square extensions. *Algorithmica*, (2) :128–138, 1999.

[11] C.M. Li and A. Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI97)*, pages 366–371, 1997.

[12] C.M. Li, W. Wei, and H. Zhang. Combining adaptive noise and look-ahead in local search for SAT. *Lecture Notes in Computer Science*, page 121, 2007.

[13] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, (6740) :133–137, 1999.

[14] MW Moskewicz, CF Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Design Automation Conference, 2001. Proceedings*, pages 530–535, 2001.

[15] S. Prestwich. Local search on SAT-encoded colouring problems. *Lecture Notes in Computer Science*, pages 105–119, 2004.

[16] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

[17] B. Selman, H.A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the national conference on artificial intelligence*, pages 337–337. JOHN WILEY & SONS LTD, 1994.

[18] T. Walsh. Sat vs csp. *Proc. CP-2000*, pages 441–456, 2000.

[19] K. Xu, F. Boussemart, F. Hemery, and C. Lecoutre. Random constraint satisfaction : Easy generation of hard (satisfiable) instances. *Artificial Intelligence*, (8-9) :514–534, 2007.

[20] K. Xu and W. Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, pages 93–103, 2000.

[21] K. Xu and W. Li. Many hard examples in exact phase transitions. *Theoretical Computer Science*, (3) :291–302, 2006.