Des chaînes d'équivalences dans un codage CNF du problème XSAT

Richard Ostrowski, Lionel Paris

Des chaînes d'équivalences dans un codage CNF du problème XSAT

Richard Ostrowski **Lionel Paris**

LSIS, Université de Provence
Marseille, France
{richard.ostrowski, lionel.paris}@cmi.univ-mrs.fr

Résumé

Étant donné une formule booléenne mise sous forme normale conjonctive (CNF), le problème de satisfiabilité XSAT, une variante du problème de satisfiabilité (SAT), consiste à trouver une interprétation des variables pour laquelle chaque clause est satisfaite par exactement un littéral. Le meilleur algorithme pour résoudre ce problème possède une complexité en temps en $(O)(2^{0.2325n})$ ($(O)(2^{0.1379n})$ pour X3SAT) [12]. Une autre possibilité pour résoudre ce problème consiste à transformer chaque clause dans un ensemble de clauses équivalentes pour le problème de satisfiabilité et d'utiliser des solveurs SAT modernes et puissants (zChaff [14], Berkmin [6], MiniSat [5], etc) pour trouver une telle interprétation. Dans ce papier, nous introduisons un nouveau codage du problème XSAT dans SAT qui contient beaucoup d'informations structurelles (en particulier les chaînes d'équivalences), qui sont cachées dans la transformation habituelle. Certains solveurs (LSAT [15], march_dl [7]) prennent en compte ce type d'informations structurelles pour faire des simplifications en prétraitement et accélérer la résolution. Puis, nous montrons l'intérêt de traiter le problème XSAT en introduisant un codage des CSP binaires et du problème de coloration de graphes sous forme de problèmes XSAT. Les résultats préliminaires sur le problème de coloration de graphes montrent l'importance du raisonnement sur les équivalences pour le problème XSAT.

Abstract

Given a boolean formula in conjunctive normal form (CNF), the Exact Satisfiability problem (XSAT), a variant of the Satisfiability problem (SAT), consists in finding an assignment to the variables such that each clause contains exactly one satisfied literal. The best algorithm to solve this problem runs in $\mathcal{O}(2^{0.2325n})$ ($\mathcal{O}(2^{0.1379n})$ for X3SAT) [12]. Another possibility is to transform each clause in a set of equivalent clauses for the Satisfiability problem and to use modern and powerful solvers (zChaff [14], Berkmin [6], MiniSat [5], etc.) to find such truth assignment. In this paper we introduce a new encoding from XSAT instances to SAT instances that leads to a lot of structural information (especially equivalencies) which is naturally hidden in the known transformation. Some solvers (lsat [15],march_dl [7]) can take into account this kind of structural information to make simplifications as pretreatment and speed-up the resolution. Then we show the interest of dealing with the XSAT formalism by introducing an encoding of binary CSP and graph coloring problem into XSAT instances. Preliminary results on graph coloring problem show the importance of equivalency reasoning for the XSAT problem.

1 Introduction

Le test de satisfaisabilité propositionnel (problème SAT) et les problèmes de satisfaction de contraintes (CSP) sont deux problèmes étroitement liés. D'une part, le problème SAT consiste à décider si une formule booléenne mise sous forme normale conjonctive (CNF) est satisfaisable. D'autre part, les CSP sont représentés par un ensemble de variables qui prennent leurs valeurs dans des domaines finis discrets et qui sont liées par des contraintes. Chaque contrainte est décrite par un tableau exprimant la relation de compatibilité entre les variables impliquées. Résoudre un CSP consiste à trouver une instanciation de toutes les variables du problème qui satisfait toutes les contraintes.

Ces deux formalismes sont largement utilisés en intelligence artificielle pour coder et résoudre des problèmes académiques comme des problèmes réels allant de la vérification à la planification. Cette utilisation a été encouragée ces dernières années par les progrès remarquables réalisés dans la résolution pratique de ces deux problèmes. L'exploitation de la structure des problèmes constitue un

facteur important de l'efficacité des solveurs SAT. Sur un grand nombre d'instances SAT, particulièrement sur celles codant des problèmes réels, l'efficacité des solveurs SAT est fortement liée à leur aptitude à exploiter les structures cachées de ces instances, comme les fonctions booléennes, les symétries, les équivalences...

Bien qu'il soit toujours théoriquement possible d'exprimer n'importe quel problème dans n'importe lequel de ces formalismes, il arrive souvent qu'un formalisme soit plus adapté que l'autre pour certains problèmes. Par exemple, il est plus simple de considérer les problèmes de coloration de graphe comme des CSP, plutôt que de les coder sous forme d'un ensemble de clauses propositionnelles. Cependant, plusieurs travaux ont présentés des transformations du formalisme CSP vers le formalisme SAT. DeKleer a tout d'abord introduit le codage direct d'un CSP en une instance SAT [9], puis Kasif a proposé le codage des supports (codage AC) [8] qui code des CSP binaires en instances SAT, permettant d'établir la propriété de consistance d'arc en utilisant la propagation unitaire. Enfin, Bessière *et al.* ont généralisé le codage des supports aux CSP non binaires en introduisant le codage k-AC [2].

Des restrictions sur la formulation du problème SAT ont conduit à un autre paradigme appelé problème XSAT. Le problème XSAT consiste à déterminer si, pour une formule sous forme CNF, il existe un modèle tel que chaque clause contient exactement un littéral satisfait.

Le problème XSAT est NP-complet, même lorsque l'on se restreint à un ensemble de clauses ternaires et où chaque variable propositionnelle n'apparaît que positivement [16, 13, 3]. Mais il n'existe pas de solveur dédié pour le problème XSAT. Quelques algorithmes ayant de bonnes complexités en temps ont été proposés pour résoudre ce problème : $\mathcal{O}(2^{0.2325n})$ et $\mathcal{O}(2^{0.1379n})$ [3, 11, 12] ; mais aucune implémentation n'a été réalisée. Généralement, les problèmes XSAT sont codés sous forme de problèmes SAT en utilisant le « codage par paires » (pairwise encoding), qui est une transformation faisant apparaître plusieurs clauses binaires négatives pour exprimer l'exclusion mutuelle entre les différents littéraux de chaque clause. Puis l'instance SAT obtenue est résolue en utilisant un solveur SAT efficace. Mais ce codage ne tient pas compte de la sémantique particulière des problèmes XSAT, et la structure cachée est perdue.

Dans cet article, nous proposons un nouveau codage des instances XSAT en SAT, qui met en évidence des informations structurelles (en particulier les chaînes d'équivalences [4]) qui pourraient être exploitées par un solveur SAT pour la résolution d'instances difficiles [15].

La motivation de l'étude du formalisme XSAT réside dans le fait qu'il permet d'exprimer beaucoup de problèmes. En particulier, nous montrons dans ce papier comment coder de manière direct les CSP binaires en utilisant des formules XSAT, ainsi que les instances du problème de coloration de graphe. Nous montrons, grâce à des résultats expérimentaux préliminaires sur certaines instances du problème de coloration de graphe que cette transformation permet de résoudre le problème original de manière efficace.

Le papier est organisé comme suit : tout d'abord, nous donnons les définitions préliminaires du contexte dans lequel se situe cette étude (SAT, XSAT et CSP). Puis nous rappelons le codage classique d'instances XSAT vers SAT et présentons notre nouveau codage qui permet d'extraire des chaînes d'équivalences. Ensuite, nous présentons le codage des CSP binaires et du problème de coloration de graphe en formules XSAT. La section 3 dresse un petit bilan des transformations connues de CSP vers SAT. La section 6 présente des résultats préliminaires obtenus pour la résolution d'instances du problème de coloration de graphe et de CSP binaires codées sous forme XSAT avant d'être codée en instances SAT en utilisant le codage décrit précédemment. Finalement, nous concluons et présentons quelques perspectives de suites à donner à notre travail.

2 Définition préliminaires

Soit $\mathcal{B}$ un langage booléen (*i.e.* propositionnel) de formules formées de manière standard, en utilisant les connecteurs usuels ($\vee$, $\wedge$, $\neg$) et un ensemble de variables propositionnelles. Une *formule CNF* Σ est un ensemble (interprété comme une conjonction) de *clauses*, où chaque clause est un ensemble (interprété comme une disjonction) de *littéraux*. Un littéral est une occurrence d'une variable propositionnelle soit sous forme positive, soit sous forme négative. À contrario, une *formule DNF* Φ est un ensemble (interprété comme une disjonction) de *termes*, où chaque terme est un ensemble (interprété comme une conjonction) de *littéraux*.

Rappelons que toute formule booléenne peut se réécrire sous forme d'une CNF en temps linéaire en utilisant la transformation de Tseitin [18]. La taille d'une CNF Σ est définie par $\sum_{c\in\Sigma} |c|$ où $|c|$ est le nombre de littéraux de c.

Une *interprétation* d'une formule booléenne est une affectation de ses variables à une valeur de vérité $\{vrai, faux\}$. Un littéral l est satisfait (resp. falsifié) pour I si l est positif et que $I[l] = vrai$ (resp. l est négatif et $I[l] = faux$). Un *modèle* pour une formule est une interprétation qui satisfait la formule. Le problème SAT est le problème de décision de la satisfaisabilité d'une CNF (existence d'un modèle).

Les chaînes d'équivalences sont de la forme $(l_1 \Leftrightarrow l_2 \Leftrightarrow \ldots \Leftrightarrow l_i)$ et sont largement étudiée dans [4]. Nous rappelons juste que l'opérateur $\Leftrightarrow$ est *commutatif* et *associatif*. Ainsi, $((l_1 \Leftrightarrow l_2) \Leftrightarrow l_3)$ et $(l_1 \Leftrightarrow (l_2 \Leftrightarrow l_3))$ sont équi-

valentes. Toutes les parenthèses peuvent être omises sans ambiguïté. De plus, deux autres règles peuvent être appliquées aux chaînes d'équivalences :

- $\neg(l_1 \Leftrightarrow l_2 \Leftrightarrow l_3)$ est équivalent à $(\neg l_1 \Leftrightarrow l_2 \Leftrightarrow l_3)$,
- les négations peuvent être supprimées par paires : $(\neg l_1 \Leftrightarrow \neg l_2 \Leftrightarrow \neg l_3)$ est équivalent à $(\neg l_1 \Leftrightarrow l_2 \Leftrightarrow l_3)$.

Si une instance contient uniquement des chaînes d'équivalences, elle peut être résolue en temps polynomial. Certains solveurs peuvent exploiter ces chaînes d'équivalences pour effectuer des simplifications en pré-traitements. Les lecteurs intéressés par ce sujet peuvent se référer à [15] pour plus de détails.

Le problème XSAT est le problème qui consiste à trouver un modèle tel qu'exactement un littéral par clause soit satisfait, et les autres doivent être falsifiés. Dans la suite, les clauses d'une instance XSAT seront appelées Xclauses pour les distinguer des clauses du problème SAT.

D'un autre côté, un CSP est un quadruplet $P = (X, D, C, R)$ où $X = \{X_1, X_2, \ldots, X_n\}$ est un ensemble de n variables, $D = \{D_1, D_2, \ldots, D_n\}$ est un ensemble de domaines finis discrets (D_i est le domaine des valeurs possibles pour X_i). $C = \{C_1, C_2, \ldots, C_m\}$ est un ensemble de m contraintes, où la contrainte C_i est définie sur un sous-ensemble de variables $\{X_{i_1}, X_{i_2}, \ldots, X_{i_{a_i}}\} \subset X$. L'arité de la contrainte C_i est a_i et $R = \{R_1, R_2, \ldots, R_m\}$ est un ensemble de m relations, où R_i est la relation correspondant à la contrainte C_i. R_i contient les combinaisons de valeurs interdites (les tuples) pour les variables impliquées dans la contrainte C_i. Un CSP binaire est un CSP dont les contraintes sont toutes d'arité deux (contraintes binaires). Un CSP est non-binaire si il contient au moins une contrainte dont l'arité est supérieure à 2 (une contrainte n-aire). Dans cet article, nous nous restreignons au cas des CSP binaires uniquement. Une *instanciation* I est une affectation qui assigne à chaque variable X_i une valeur de son domaine D_i. Une contrainte C_i est satisfaite par une instanciation I si la projection de I sur les variables impliquées dans C_i diffère de tous les tuples de la relation R_i. Une instanciation I d'un CSP P est consistante (ou est une solution de P) si elle satisfait toutes les contraintes de P. Un CSP P est consistant si il admet au moins une solution ; sinon il est inconsistant. SAT et CSP sont tous deux des problèmes NP-complet. Dans la suite du papier, nous considérons que n est le nombre de variables du CSP, m est son nombre de contraintes, a l'arité maximale de ses contraintes et d la taille du plus grand de ses domaines.

3 Coder les problèmes XSAT en instances SAT

Dans cette section nous rappelons le codage basique d'une Xclause sous la forme d'un ensemble de clauses équivalentes pour le test de satisfaisabilité. Puis nous introduisons le nouveau codage de XSAT vers SAT que nous appelons codage par ajout de chaînes d'équivalences.

3.1 Codage basique

Convertir une Xclause en un ensemble de clauses peut être fait en introduisant une clause contenant les même littéraux que la Xclause, en combinaison avec un ensemble de clauses binaires négatives exprimant l'exclusion mutuelle entre chaque couple de littéraux. Étant donné la Xclause $c = l_1 \vee l_2 \vee ... \vee l_n$, l'ensemble de clauses correspondant à son codage SAT est la clause $c' = l_1 \vee l_2 \vee ... \vee l_n$ et l'ensemble de clauses binaires :

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n} \neg l_i \vee \neg l_j.$$

Coder une Xclause contenant n littéraux de cette manière se fait avec une complexité en espace en $O(n^2)$. Ce codage porte le nom de « codage par paires » (pairwise encoding).

3.2 Codage par ajout de chaînes d'équivalences

Dans cette section, nous décrivons le nouveau codage d'instances XSAT en instances SAT dans lequel il est possible de trouver des chaînes d'équivalences. Certains solveurs (eqsatz [10], lsat [15], march_dl [7]) utilisent des raisonnement sur les équivalences pour faire des simplifications ou pour aider les heuristiques à choisir de meilleures variables.

Si nous considérons la Xclause $Xc = l_1 \vee l_2 \vee ... \vee l_n$, nous avons vu dans la section précédente qu'il est possible de la coder sous la forme d'un ensemble de clauses SAT équivalent pour la satisfaisabilité.

Maintenant, parmi toutes les 2^n interprétations possibles sur les n variables de Xc, seulement n satisfont la propriété *exactement un littéral satisfait* (par rapport à Xc). Considérons toutes les $2^n - n$ autres (qui ne satisfont pas la propriété). Nous pouvons les considérer comme une DNF contenant $2^n - n$ termes. Un moyen simple d'obtenir une formule vérifiant la propriété est de prendre la formule complémentaire de cette DNF. Et la complémentaire d'une DNF est une CNF, ainsi nous obtenons une CNF contenant $2^n - n$ clauses qui codent la Xclause Xc. De cette CNF, il peut aisément être extraite une équivalence entre les littéraux apparaissant dans la Xclause plus quelques autres clauses. En fait, les modèles d'une chaîne d'équivalences $(l_1 \leftrightarrow l_2 \leftrightarrow ... \leftrightarrow l_n)$ vérifient la propriété d'avoir un nombre pair ou impair de littéraux satisfaits en fonction de la longueur de la chaîne considérée.

Voici deux exemples simples pour illustrer le processus.

Exemple 1 *Soit la Xclause* $Xc = l_1 \vee l_2$. *Les termes* $\{m_1 = (l_1 \wedge l_2), m_2 = (\neg l_1 \wedge \neg l_2)\}$ *sont les seuls qui*

ne respectent pas la propriété exactement un littéral satisfait par rapport à Xc. En prenant le complémentaire de ces clauses, nous obtenons la formule CNF $\{c_1 = (\neg l_1 \vee \neg l_2), c_2 = (l_1 \vee l_2)\}$. Maintenant, de cet ensemble de clauses, il est simple d'extraire la chaîne d'équivalences $(\neg l_1 \leftrightarrow l_2)$.

Exemple 2 *Soit la Xclause $Xc = l_1 \vee l_2 \vee l_3$ contenant 3 littéraux. Les termes $\{m_1 = (\neg l_1 \wedge \neg l_2 \wedge \neg l_3), m_2 = (\neg l_1 \wedge l_2 \wedge l_3), m_3 = (l_1 \wedge \neg l_2 \wedge l_3), m_4 = (l_1 \wedge l_2 \wedge \neg l_3), m_5 = (l_1 \wedge l_2 \wedge l_3)\}$ sont les seuls à ne pas respecter la propriété. En prenant leurs complémentaires, nous obtenons la formule CNF $\{c_1 = (l_1 \vee l_2 \vee l_3), c_2 = (l_1 \vee \neg l_2 \vee \neg l_3), c_3 = (\neg l_1 \vee l_2 \vee \neg l_3), c_4 = (\neg l_1 \vee \neg l_2 \vee l_3), c_5 = (\neg l_1 \vee \neg l_2 \vee \neg l_3)\}$. De cet ensemble de clauses, en considérant les clauses c_1, c_2, c_3, c_4, la chaîne $l_1 \leftrightarrow l_2 \leftrightarrow l_3$ peut être extraite. La clause C_5 empêchant la chaîne d'équivalences d'avoir trois littéraux à vrai dans un modèle.*

Plus généralement, si nous avons $Xc = l_1 \vee l_2 \vee ... \vee l_n$, nous avons 2^{n-1} clauses qui représentent la chaîne d'équivalences plus $2^{n-1} - n$ clauses additionnelles pour empêcher la chaîne d'avoir plus d'un littéral à vrai.

Remarque 1 *Remarquons qu'il est possible d'obtenir le codage par paires décrit précédemment à partir du codage par ajout de chaînes d'équivalences en utilisant les règles de résolution et de subsumption propositionnelle.*

3.3 Réduction de la complexité du codage par ajout de chaînes d'équivalences

La complexité de ce codage représente malheureusement son principal défaut. Nous avons vu que pour chaque Xclause de longueur n, nous avons besoin d'un ensemble de $2^n - n$ clauses. La taille du codage SAT croît de manière exponentielle par rapport à la taille de l'instance XSAT, et devient rapidement intraitable.

Heureusement, il est possible de réduire drastiquement la taille de la formule codée, en introduisant de nouvelles variables supplémentaires.

Proposition 1 *Il est possible de convertir une Xclause de longueur n en un ensemble de $n-2$ clauses de longueur 3. Notons que cette conversion introduit $n-3$ extra variables.*

Preuve 1 *Nous ne donnons que l'idée générale de la preuve. Elle est basée sur une preuve par induction sur la longueur de la Xclause et sur le fait qu'une Xclause $Xc = l_1 \vee l_2 \vee ... \vee l_n$ est équivalente pour la satisfaisabilité à $(l_1 \vee l_2 \vee x_1) \wedge (\neg x_1 \vee l_3 \vee ... \vee l_n)$.*

Ainsi, étant donné qu'une Xclause de longueur 3 est codée avec 5 clauses de longueur 3 en utilisant le codage par ajout de chaînes d'équivalences, une Xclause de longueur n peut être codée avec $5(n-2)$ clauses de longueur 3.

4 Transformation XSAT d'un CSP binaire

Dans cette partie, nous présentons comment il est possible de transformer un CSP binaire en un problème XSAT équivalent. Étant donné un CSP $P = (X, D, C, R)$, nous définissons, dans un premier temps, l'ensemble des variables propositionnelles que nous utiliserons dans la transformation en problème XSAT. Nous considérons ensuite deux types de XClauses : celles qui représentent les domaines des variables du problème en CSP et celles qui représentent les contraintes du problème.

- L'ensemble des variables booléennes : tout comme dans les codages existants [9], [8] et [2], nous associons pour chaque variable X et pour chaque valeur v de cette variable, une variable booléenne X_v. X_v = *vrai* signifie que la variable X à la valeur v dans le CSP. Ainsi, nous avons besoin de $\sum_{i=1}^{n} |D_i|$ variables booléennes. Le nombre de variables booléennes est d'au maximum nd.
- Les Xclauses pour le domaine de chaque variable : chaque variable du CSP ne peut prendre qu'une seule valeur dans son domaine. Dans la transformation en problème XSAT, ces contraintes sont représentées par une seule Xclause contenant les littéraux associés aux valeurs des domaines. Soit X une variable du CSP et $D_X = \{v_0, v_1, \ldots, v_k\}$ son domaine. La Xclause pour coder le domaine est : $Xc_X = X_{v_0} \vee X_{v_0} \vee \ldots \vee X_{v_k}$.
  Nous avons besoin de n Xclauses pour représenter les contraintes de domaine sur les variables.
- Les Xclauses pour les contraintes du CSP : pour chaque contrainte, nous avons autant de Xclauses qu'il y a de tuples interdits. Ces Xclauses contiennent les deux littéraux positifs correspondant au couple interdit de la contrainte considérée plus une variable additionnelle. Nous devons ajouter autant de variables qu'il y a de tuples interdits. Soit X et Y deux variables du CSP et (v_i, v_j), $(v_i \in D_X, v_j \in D_Y)$ un couple interdit. Nous produisont la Xclause $Xc_c = X_{v_i} \vee Y_{v_j} \vee t_k$ où t_k représente la variable additionnelle.

  Dans le pire cas, nous avons besoin de d^2 Xclauses pour exprimer une contrainte du CSP.

Nous traitons à présent de la correction et de la complétude de la transformation des CSP binaires vers XSAT.

Définition 1 *Soit P un CSP binaire et P_{Xsat} sa transformation en problème XSAT. Soit I une instanciation de P. L'interprétation équivalente I_{Xsat} de P_{Xsat} doit vérifier la condition suivante : pour chaque variable X du CSP, pour chaque valeur v_i de son domaine, $I_{\mathrm{Xsat}}[X_{v_i}]$ =vrai si et seulement si $I[X] = v_i$.*

Proposition 2 *Soit P un CSP, P_{Xsat} sa transformation en problème XSAT, I une instanciation de P et I_{Xsat} son interprétation équivalente pour le problème P_{Xsat}. I est une solution de P si et seulement si I_{Xsat} est un modèle de P_{Xsat}.*

Preuve 2 *Soit I_{Xsat} un modèle de P_{Xsat} et I l'instaciation correspondante pour P. I_{Xsat} satisfait chaque Xclause de P_{Xsat}, car c'est un de ses modèles. Nous devons montrer que chaque variable du CSP n'a qu'une seule valeur dans I et que I satisfait toutes les contraintes de P. Pour chaque variable de domaine de P, nous introduisons une Xclause exprimant le fait de n'avoir qu'une seule valeur par variable. Comme chaque I_{Xsat} les satisfait toutes, nous avons bien le fait que chaque variable n'a qu'une seule valeur. D'autre part, les autres Xclauses, pour les contraintes sur les tuples interdits, sont aussi unisatisfaites. Cela signifie que pour chaque Xclause $c_c = X_{v_i} \vee Y_{v_j} \vee t_k$ soit X_{v_i}, Y_{v_j} ou t_k a la valeur de vérité $vrai$. Si on considère le tuple interdit ($X = v_i, Y = v_j$), soit $X = v_i$, soit $Y = v_j$ ou aucun des deux. Nous ne pouvons jamais avoir les deux à $vrai$. Ceci implique qu'aucune contrainte du CSP n'est violée et I est donc une solution de celui ci. La réciproque se montre de la même façon.*

Nous donnons à présent un petit exemple afin d'illustrer cette transformation.

Exemple 3 *Considérons un CSP binaire contenant deux variables X et Y avec pour domaine respectif $D_X = \{a, b, c\}$ et $D_Y = \{d, e, f\}$. Une contrainte entre ces deux variables est représentée par la relation listant les couples interdits :*

R_{XY}	
X	Y
a	f
b	e
c	d
c	f

La transformation de ce problème en un problème XSAT se compose des Xclauses suivantes :
$Xc_X = X_a \vee X_b \vee X_c$
$Xc_Y = Y_d \vee Y_e \vee Y_f$
$Xc_{XY_1} = X_a \vee Y_f \vee t_1$
$Xc_{XY_2} = X_b \vee Y_e \vee t_2$
$Xc_{XY_3} = X_c \vee Y_d \vee t_3$
$Xc_{XY_4} = X_c \vee Y_f \vee t_4$

4.1 Complexité de la transformation de CSP vers XSAT

Comme nous l'avons vu, nous avons besoin de n Xclauses de taille d afin d'exprimer les n variables du CSP ayant pour taille de domaine d. Nous avons aussi besoin de m ensembles de Xclauses de taille 3 pour coder les m contraintes du CSP. Chaque ensemble peut comporter d^2 Xclauses (dans le cas où tous les couples sont autorisés). Dans ce cas la complexité en espace est en $\mathcal{O}(nd + md^2) = \mathcal{O}(md^2)$.

4.2 Codage XSAT d'un problème de coloration de graphe

Nous avons décidé d'expérimenter notre codage pour la résolution de problèmes CSP. Nous nous sommes penchés sur le problème de coloration de graphe. Nous présentons ici la façon dont nous transformons un problème de k-coloration de graphe[1] en un problème XSAT.

Un problème de k-coloration de graphe se définit de la façon suivante : étant donné un graphe non-orienté $G = (V, E)$, le problème consiste à trouver une coloraton du graphe en utilisant k couleurs de telle sorte que chaque sommet de G ait une couleur différente de ses voisins. La k-coloration de graphe est dans P pour $k \leq 2$ et NP-complet pour $k \geq 3$. Un problème proche de la k-coloration de graphe est le problème de déterminer le plus petit k (nombre chromatique) permettant de colorer le graphe.

Ici, nous nous intéressons au problème de décision, c'est-à-dire, étant donné un graphe non orienté $G = (V, E)$ et k un nombre de couleurs autorisées, déterminer si G admet une k-coloration. Pour transformer un problème de coloration de graphe en un problème XSAT, nous devons considérer, tout comme pour les CSP, deux sortes de contraintes :

- Chaque sommet $V_1, ..., V_n$ de G n'a qu'une seule couleur $C_1, ...C_k$ (domaine des variables du CSP) :

$$V_iC_1 \vee V_iC_2 \vee ... \vee V_iC_k \text{ avec } i \in \{1, ..., n\}.$$

  $V_iC_j (i \in \{1, ..., n\}, j \in \{1, ..., k\})$ représente une variable propositionnelle indiquant que le sommet i est de couleur j.
- Considérant une arête $e \in E$ entre les sommets V_1 et V_2 (une contrainte du CSP), si V_1 est de couleur l alors V_2 ne peut pas être de la même couleur : $V_iC_l \vee V_jC_l \vee T_{(i,j,l)}$. Tout comme la transformation d'un CSP vers un problème XSAT, nous avons besoin d'ajouter de nouvelles variables propositionnelles ($T_{(i,j,l)}$, $i, j \in \{1, ..., n\}, l \in \{1, ..., k\}$). Elles sont ajoutées afin de garantir que tout modèle du problème de coloration de graphe est un modèle du problème XSAT.

Exemple 4 *Soit G=(V,E) un graphe non-orienté, $V = \{V_1, V_2, V_3\}$, $E = \{(V_1, V_2), (V_2, V_3), (V_1, V_3)\}$ et $k = 3$. Le problème XSAT équivalent est le suivant :*

[1] la k-coloration de graphe consiste à déterminer si un graphe peut se colorer en utilisant k couleurs

Une couleur par sommet :	*Pour tout* $e \in E$ *:*
$(V_1C_1 \vee V_1C_2 \vee V_1C_3)$	$(V_1C_1 \vee V_2C_1 \vee T_1)$
$(V_2C_1 \vee V_2C_2 \vee V_2C_3)$	$(V_1C_2 \vee V_2C_2 \vee T_2)$
$(V_3C_1 \vee V_3C_2 \vee V_3C_3)$	$(V_1C_3 \vee V_2C_3 \vee T_3)$
	$(V_2C_1 \vee V_3C_1 \vee T_4)$
	$(V_2C_2 \vee V_3C_2 \vee T_5)$
	$(V_2C_3 \vee V_3C_3 \vee T_6)$
	$(V_1C_1 \vee V_3C_1 \vee T_7)$
	$(V_1C_2 \vee V_3C_2 \vee T_8)$
	$(V_1C_3 \vee V_3C_3 \vee T_9)$

On peut facilement voir que I*={*V_1C_1*,* V_2C_2*,* V_3C_3*,* $\neg V_1C_2$*,* $\neg V_1C_3$*,* $\neg V_2C_1$*,* $\neg V_2C_3$*,* $\neg V_3C_1$*,* $\neg V_3C_2$*,* $\neg T_1$*,* $\neg T_2$*,* T_3*,* T_4*,* $\neg T_5$*,* $\neg T_6$*,* $\neg T_7$*,* T_8*,* $\neg T_9$*} est un modèle pour XSAT et un modèle pour le problème de* 3*-coloration considéré.*

4.3 Complexité du codage vers SAT d'un CSP utilisant les équivalences

Nous avons vu que dans le pire des cas, la complexité de la transformation d'un CSP binaire en un problème XSAT est de $(nd + 3md^2)$. Ce problème XSAT contient md^2 Xclauses de taille 3 et n Xclauses de taille d. Lorsque nous transformons le problème XSAT en un problème SAT utilisant les chaînes d'équivalences, nous obtenons $(n(2^d - d))$ clauses de taille d et $(5md^2)$ clauses de taille 3. Maintenant, si nous transformons chaque Xclause de taille strictement supérieure à 3 en des Xclauses de taille 3, nous obtenons $(5(n(d-2) + md^2)$ clauses de longueur 3 une fois transformé en SAT. Cette complexité est comparable avec le codage direct qui a une complexité en $\mathcal{O}((n+m)d^2)$.

Dans le cas d'un CSP représentant un problème de k-coloration (avec n sommets et m arêtes), la complexité de la transformation en SAT est de $(n(2^k - k))$ clauses de longueur k et de $(5mk)$ clauses de longueur 3. Dans le cas où les Xclauses ont été raccourcies, cette complexité est de $(5(n(k-2) + mk)$ clauses de longueur 3.

5 Travaux de référence

L'idée de transformer un CSP, et plus généralement des contraintes, en une formule SAT équivalente n'est pas nouvelle et à été largement étudiée. D'abord , De Kleer dans [9] à introduit *le codage direct* qui transforme un CSP en un ensemble de clauses. Ensuite, Kasif [8] a proposé *le codage de supports* qui est une amélioration du codage direct mais pour les CSP binaires. Dans cette transformation, la propagation unitaire permet de restaurer la consistance d'arc en complexité linéaire. Plus récemment, Bessière et al [2] ont généralisé le codage des supports aux CSP n-aire.

Un codage spécifique des contraintes de cardinalités a été introduit par Bailleux et Boufkhad dans [1]. La propagation unitaire sur ce codage permet le maintient de l'arc consistance généralisée. Dernièrement, Marques-Silva et Lynce [17] ont étudié les différents codages possibles des contraintes de cardinalité dans SAT et ont montré que ceux nécessitant des variables additionnelles peuvent être ignorés par les solveurs afin de rendre la résolution plus robuste.

D'àprès nos connaissances, bien que de nombreux travaux antérieurs ont abordé le problème de transformation d'un problème CSP vers un problème SAT équivalent à différents points de vue, ce travail est le premier introduisant des équivalences dans la transformation.

6 Expérimentations

Dans cette partie, nous présentons les résultats expérimentaux que nous avons obtenu sur des problèmes de coloration de graphe (problème de décision) issues des challenges DIMACS (`http://mat.gsia.cmu.edu/COLOR04/`), et sur des CSP binaires générés aléatoirement.

6.1 Problèmes de coloration de graphe

Nous avons essayé de trouver une k-coloration pour différents problèmes avec différentes valeurs pour k (de 2 à 7). Pour chaque graphe, et chaque valeur de k, nous avons dans un premier temps transformé le problème de coloration de graphe en un problème XSAT équivalent. Ensuite nous avons transformé ce problème XSAT en un problème SAT équivalent en utilisant le codage par paire (pairwise encoding PWE) et celui introduisant les chaînes d'équivalences (EE) sans couper les Xclauses de taille strictement supérieure à 3. Cela est possible étant donné que nous avons testé de petites valeurs pour k. Nous n'avons pas mis les résultats lorsque que l'on coupe ces Xclauses car les résultats ne sont pas concluant étant donné que l'on perd de la sémantique lorsque l'on ajoute les variables additionnelles.

Pour chaque problème, nous comparons le temps CPU (en seconde) de trois solveurs (zChaff, March_dl et eqsatz) avec un timeout de 1000 secondes. Les expérimentations ont tourné sur un P IV 3.0 Ghz avec 1 Go de RAM. Le tableau 1 présente les résultats. La première colonne donne le nom de l'instance, la colonne k donne le nombre de couleurs autorisées. Enfin les colonnes zChaff, march_dl et eqsatz donnent le temps CPU pour résoudre les instances pour chaque codage (PWE et EE).

Même si les premiers résultats sont de manière générale encourageants, nous espérions de bien meilleurs résultats de march_dl et d' eqsatz que zChaff. En effet ces deux solveurs sont assez spécialisés dans la recherche et la simplification d'équivalences au cours de la recherche. Pour eqsatz, cela peut en partie s'expliquer par le fait que la détection des équivalences de taille strictement supérieure à 3 ne se fait que tardivement au cours de la recherche.

Instance		zChaff		march_dl		eqsatz	
	k	PWE (s)	EE (s)	PWE (s)	EE (s)	PWE (s)	EE (s)
fpsol2.i.1	2	1.02	0.39	3.61	2.67	0.47	1.34
	3	5.83	0.58	22.76	3.39	3.61	2.67
	4	33.68	0.81	63.22	16.75	5.31	60.5
	5	96.84	1.19	53.74	90.25	214	358
	6	77.73	1.36	67.75	—	—	—
	7	61.5	2.11	—	—	—	—
inithx.i.1	2	4.89	0.62	0.38	5.13	0.76	13.8
	3	41.14	0.99	172.8	6.48	—	—
	4	164.8	1.33	80.15	26.65	—	—
	5	143.5	1.76	75.21	129	—	81
	6	77.7	2.35	104.9	510.7	—	97
	7	12.51	3.54	—	—	—	1.93
latin_square_10	2	—	10.55	—	491.5	4.41	6.8
	3	—	15.89	353.5	—	5.84	7.36
	4	—	21.29	—	—	5.93	7.21
	5	—	27.09	13.03	—	5.93	7.27
	6	—	34.51	—	—	5.88	7.54
	7	—	44.77	—	—	5.93	7.62
le450_5a	2	0.11	0.24	2.95	0.61	4.17	6.95
	3	0.17	0.35	1.65	0.59	100.5	122
	4	0.24	1.36	5.26	15.66	—	216
	5	4.85	—	128.7	—	—	—
	6	—	—	366.7	—	—	—
	7	—	—	543.9	—	—	—
miles1500	2	1.88	0.39	2.65	3.9	4.27	55
	3	15.15	0.56	100.3	1.57	185	164
	4	38.55	0.74	86.05	4.16	—	—
	5	198	0.95	808.5	83.68	—	—
	6	318.76	1.22	138.1	—	—	139
	7	179.51	1.81	789.4	—	—	133
mulsol.i.5	2	0.11	0.17	0.38	0.53	2.31	4.22
	3	0.33	0.24	17.57	0.57	5.03	8.25
	4	0.85	0.33	6.05	1.62	30.6	38.5
	5	1.63	0.41	31.36	24.55	—	—
	6	2.58	0.53	815.9	—	—	—
	7	6.17	0.85	—	—	—	—
myciel6	2	0.01	0.06	0.10	0.06	0.45	0.94
	3	0.06	0.07	0.35	0.13	1.95	2.23
	4	1.49	2.15	4.17	9.5	35.2	36.2
	5	—	—	680.9	—	—	—
	6	—	—	—	—	—	—
	7	0.08	—	3.5	1.98	20.8	4.35
queen12_12	2	0.1	0.22	0.63	1.07	2.34	4.22
	3	0.16	0.31	2.88	0.56	7.45	12.3
	4	0.24	0.41	5.53	1.95	38.2	33.1
	5	0.4	0.63	56.85	642.4	—	172
	6	0.77	19.81	29.13	—	—	—
	7	1.47	953.3	—	—	—	—
school1_nsh	2	1.25	0.55	12.55	2.84	2.47	14.9
	3	9.34	0.77	36.11	3.33	4.91	68.5
	4	10.33	1.05	29.73	9.63	242	296
	5	15.39	1.34	80.56	38.74	—	—
	6	16.96	1.76	72.94	747.8	—	—
	7	4.1	11.18	—	—	—	130
school1	2	3.77	0.73	27.53	4.47	4.17	6.95
	3	27.14	1.02	367	5.72	8.37	123
	4	77.07	1.61	115.5	23.6	—	—
	5	16.17	1.76	91.56	312.7	—	76
	6	8.15	2.32	168.7	—	—	93
	7	12.48	3.78	—	—	—	45
zeroin.i.1	2	0.16	0.19	0.5	0.61	6.21	10.4
	3	0.75	0.25	12.76	0.65	26.7	34.6
	4	2.71	0.32	16.1	1.93	318	186
	5	8.89	0.41	92.9	39	—	435
	6	21.68	0.55	10.3	814.8	—	—
	7	4.72	0.88	—	—	—	—

TAB. 1 – Performances des solveurs sur des problèmes de coloration de graphe

D'autre part, les bonnes performances de zChaff en utilisant le codage par ajout de chaînes d'équivalences sont surprenants mais intéressants. Ceci montre que la transformation par ajout d'équivalences peut dans certains cas compenser la perte des clauses binaires en considérant le codage par paire. D'àprès nous, zChaff ne tient pas compte des chaînes d'équivalences au cours de la résolution (à moins que la dernière version le fasse). Cela peut signifier que notre codage possède des informations intéressantes pour zChaff.

6.2 CSP binaires aléatoires

Nous avons ensuite testé les performances des trois solveurs et de notre transformation sur des problèmes CSP binaires aléatoires. Chaque ligne du tableau 2 donne la moyenne des résultats obtenus sur 20 problèmes différents. La première colonne donne le détail du problème traité : son nombre de variables (n), la taille des domaines (D), la densité du graphe ($Dens$) et la dureté moyenne des contraintes ($Tight$). Pour chaque solveur, nous donnons le temps de résolution (en seconde) en considérant le codage direct du CSP (DE), le codage par supports (SE) et le codage par chaînes d'équivalences (EE).

Comme pour les problèmes de coloration de graphe, nous ne mettons pas les résultats lorsque l'on coupe les Xclauses de taille supérieure à 3.

Les résultats obtenus ne sont pas trop concluant sur ces types de problèmes. Notre codage donne de moins bons résultats pour les 3 solveurs. Ceci peut en partie s'expliquer par le fait qu'il n'y a pas réellement de structure dans les CSP binaires aléatoires. D'autre part la taille de la formule vis à vis de notre transformation est plus importante que les deux autres codages et les solveurs doivent les traiter sans exploiter aucune structure, ce qui rend la résolution plus longue de manière générale.

Remarque 2 *Afin de rendre plus efficace la coupure des Xclauses de taille supérieure à 3 par ajout de variables additionnelles, nous devons forcer les solveurs à ne pas tenir compte de ces nouvelles variables dans le choix des variables à affecter au cours de la recherche. Par manque de temps, nous n'avons pas pu tester cette possibilité.*

Une autre possibilité est d'avoir une formule mixte contenant à la fois la définition concise des chaînes d'équivalences et une partie CNF. Il faut pour cela avoir un solveur hybride qui tiendrait compte de cette formule mixte.

Cependant, cette nouvelle transformation ne semble pas intéressante pour les CSP non structurés (comme les CSP aléatoires). Des expérimentations sur des CSP binaires représentant des problèmes réels devraient donner d'aussi bons résultats que sur les problèmes de coloration de graphe (qui ne sont en fait que des CSP binaires particuliers).

7 Conclusions et perspectives

Dans ce papier, nous avons proposé une nouvelle façon de coder les CSP binaires et les problèmes de coloration de graphe sous la forme d'instance SAT. Ce codage se fait en deux étapes. Tout d'abord nous codons le problème de départ sous la forme d'une formule XSAT, puis nous codons la formule XSAT en un problème SAT. Contrairement aux codages existants (direct, support...), notre nouveau codage ajoute des informations structurelles concernant des chaînes d'équivalences dans le problème SAT. Ces informations sont utilisées par des solveurs comme prétraitements pour simplifier la formule. Les performances du solveur zChaff sur des instances difficile du problème de coloration de graphe montre que ces informations sont pertinentes et peuvent améliorer significativement les performances de résolution.

Ce travail ouvre plusieurs perspectives intéressantes. Tout d'abords, il serait intéressant de trouver un moyen de coder les CSP non binaires sous forme de formules XSAT, pour pouvoir les transformer en instances SAT ensuite en utilisant ce nouveau codage. Dans un second temps, cela vaudrait la peine d'étudier ce que deviennent les propriétés comme les consistances locales lors des transformations de CSP vers SAT (en passant par XSAT), de comparer la propagation unitaire dans le problème SAT résultant avec des algorithmes de filtrages par consistances locales dans le CSP de départ.

Références

[1] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *CP*, pages 108–122, 2003.

[2] C. Bessière, E. Hebrard, and T. Walsh. Local consistency in SAT. In *International Conference on Theory and Application of satisfiability testing*, pages 400–407, 2003.

[3] Vilhelm Dahllöf. Applications of general exact satisfiability in propositional logic modelling. In *LPAR*, pages 95–109, 2004.

[4] B. Dunham and H. Wang. Towards feasible solutions of the tautology problem. *Annals of Math. Log.*, 10 :117–154, 1976.

[5] Niklas Eén and Niklas Sörensson. An extensible satsolver. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, pages 502–518, 2003.

[6] Robert K. Brayton Evguenii I. Goldberg, Mukul R. Prasad. Using problem symmetry in search based satisfiability algorithms. In *proceedings of the Design, Automation and Test in Europe Conference and*

Instance				zChaff			march_dl			eqsatz		
n	D	$Dens$	$Tight$	DE (s)	SE (s)	EE (s)	DE (s)	SE (s)	EE (s)	DE (s)	SE (s)	EE (s)
50	4	0.2	0.1	0.2	0.3	0.5	0.3	0.8	0.9	1.3	1.5	1.7
			0.2	0.3	0.3	0.8	0.5	3.5	1.2	1.3	1.6	2.1
			0.3	0.3	0.3	0.9	0.3	0.4	1.2	0.8	1.1	2.2
			0.4	0.3	0.3	0.8	0.3	0.3	1.3	0.8	0.9	2.5
			0.5	0.3	0.3	0.7	0.3	0.3	1.3	0.8	0.8	2.5
			0.6	0.3	0.3	0.8	0.3	0.3	1.3	0.8	0.7	2.7
			0.7	0.3	0.3	0.9	0.3	0.2	1.3	0.9	0.7	3.1
			0.8	0.3	0.3	0.9	0.3	0.2	1.6	0.9	0.6	3.2
			0.9	0.3	0.3	0.9	0.3	0.2	1.7	1	0.6	3.2
50	4	0.5	0.1	0.3	0.3	0.6	0.4	1.6	1.1	1.3	1.9	2.6
			0.2	0.3	0.3	0.8	0.3	0.5	2.1	0.9	1.6	2.8
			0.3	0.3	0.3	0.9	0.3	0.4	1.9	0.9	1.9	3.1
			0.4	0.3	0.4	1.1	0.3	0.3	2.2	1	1.5	3.3
			0.5	0.4	0.3	1.1	0.3	0.3	2.3	1	1.1	3.5
			0.6	0.4	0.4	1.4	0.4	0.3	3.1	1.1	0.9	3.8
			0.7	0.4	0.4	1.4	0.4	0.3	3.6	1.3	0.7	4.1
			0.8	0.4	0.4	1.5	0.4	0.3	3.8	1.4	0.7	4.3
			0.9	0.4	0.4	1.5	0.4	0.3	4.2	1.4	0.7	4.7
50	4	0.9	0.1	1.8	5.8	6.3	0.6	3.6	22	1.8	0.8	13.1
			0.2	0.4	0.7	1.1	0.3	0.6	4.2	0.9	2.9	6.7
			0.3	0.4	0.7	1.3	0.3	0.5	5.3	1	4.2	10.7
			0.4	0.4	0.6	1.5	0.4	0.4	10.1	1.3	2.2	11.3
			0.5	0.4	0.6	1.5	0.4	0.4	10.3	1.4	1.4	14.4
			0.6	0.6	0.5	1.7	0.5	0.4	10.3	1.8	0.8	19.3
			0.7	0.9	0.5	1.8	0.5	0.4	11.2	2	0.8	25.5
			0.8	0.6	0.4	2.1	0.8	0.3	11.8	2.5	0.7	32.2
			0.9	0.6	0.4	2.3	0.6	0.3	13.2	2.8	0.8	33.7
80	4	0.2	0.1	0.3	0.7	0.6	0.4	1.6	1.5	1.5	2	3.6
			0.2	0.4	0.6	1.6	0.4	0.5	6.2	1.2	1.7	6.1
			0.3	0.4	0.4	1.1	0.4	0.5	2.8	0.9	1.5	3.6
			0.4	0.4	0.4	1.2	0.4	0.4	2.5	1	1.4	5.8
			0.5	0.4	0.4	1.1	0.4	0.4	2.9	1	1.3	7.4
			0.6	0.4	0.4	1.4	0.4	0.3	3.5	1	0.8	10.1
			0.7	0.4	0.4	1.5	0.4	0.3	4.1	1.1	0.9	12.5
			0.8	0.4	0.4	1.6	0.4	0.3	4.2	1.2	0.7	14.2
			0.9	0.4	0.4	1.6	0.4	0.3	4.8	1.4	0.7	16.6
80	4	0.5	0.1	14.3	16.89	17.42	2.2	10.2	36.8	20.3	16.4	46.2
			0.2	0.4	0.7	1.4	0.4	0.8	5.2	1.1	3.8	9.1
			0.3	0.4	0.7	1.5	0.4	0.7	6.4	1.2	5.9	9.4
			0.4	0.5	0.7	1.9	0.5	0.4	6.8	1.4	2.9	10.3
			0.5	0.5	0.6	2.3	0.6	0.4	7.1	1.5	1.8	12.6
			0.6	0.6	0.6	2.5	0.7	0.4	7.5	1.7	0.9	14.8
			0.7	0.6	0.6	2.4	0.8	0.5	7.8	2.4	0.9	16.8
			0.8	0.8	0.6	2.7	0.8	0.4	8.1	2.7	0.8	17.6
			0.9	0.8	0.6	2.9	0.9	0.4	8.7	3.2	0.8	25.7
80	4	0.9	0.1	1.3	5.3	8.1	0.6	10.6	9.4	2	15.5	8.8
			0.2	0.4	1.1	1.9	0.5	1.2	1.7	1.4	9.8	2.8
			0.3	0.6	1.1	2.4	0.6	1.1	3.2	1.7	15.7	4.7
			0.4	0.7	1	3.2	0.7	0.6	6.3	2.8	4.6	9.8
			0.5	0.8	1.1	3.9	0.8	0.5	5.6	3.4	2.2	12.7
			0.6	0.8	1	4.6	0.9	0.5	5.7	4.9	1.2	20.1
			0.7	0.9	0.8	5	0.9	0.5	6.2	5.7	1.1	24.2
			0.8	1	0.8	5.8	1	0.5	6.8	7.8	1	33.5
			0.9	1.1	0.7	6.1	1.1	0.6	7.7	8.9	0.9	38.2

TAB. 2 – Performances des solveurs sur des CSP aléatoires

Exhibition (DATE'02), pages 134–141. IEEE Computer Society, 2002.

[7] Marijn Heule, Joris van Zwieten, Mark Dufour, and Hans van Maaren. March_eq : Implementing additional reasoning into an efficient lookahead sat solver. In *Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT2004)*, pages 345–359, 2004.

[8] S. Kasif. On the Parallel Complexity of Discrete Relaxation in Constraint Satisafaction Networks. *Journal of Artificial Intelligence*, 45 :275–286, 1990.

[9] J. De Kleer. A comparison of ATMS and CSP techniques. In *Proceedings of IJCAI'89*, pages 290–296, DETROIT, 1989.

[10] Chu-Min Li. Equivalent literal propagation in the dll procedure. *Discrete Appl. Math.*, 130(2) :251–276, 2003.

[11] B.A. Madsen, J.M. Nielsen, and B. Skjernaa. New algorithms for exact satisfiability. Technical Report RS-03-30, Aarhus University, 2003.

[12] Bolette Ammitzböll Madsen. An algorithm for exact satisfiability analysed with the number of clauses as parameter. *Inf. Process. Lett.*, 97(1) :28–30, 2006.

[13] B. Monien, E. Speckenmeyer, and O. Vornberger. Upper bound for covering problems. *Methods of Operations Research*, 34 :419–431, 1981.

[14] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC01)*, 2001.

[15] R. Ostrowski, E. Grégoire, B. Mazure, and L. Saïs. Recovering and exploiting structural knowledge from cnf formulas. In *CP'02*, pages 185–199, 2002.

[16] Thomas J. Schaefer. The complexity of satisfiability problems. In *STOC*, pages 216–226, 1978.

[17] João P. Marques Silva and Inês Lynce. Towards robust cnf encodings of cardinality constraints. In *CP*, pages 483–497, 2007.

[18] G.S. Tseitin. On the complexity of derivations in the propositional calculus. In H.A.O. Slesenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.