Formules CNF et Graphes

Said Jabbour

CRIL-CNRS, Université d'Artois, 62307 Lens Cedex, France
jabbour@cril.univ-artois.fr

Résumé :
Nous proposons dans cet article une nouvelle représentation sous forme de graphe d'une formule CNF. Elle étend le graphe d'implication 2-SAT au cas général. Chaque clause est représentée comme un ensemble (conditionnel) d'implications et codée avec plusieurs arcs étiquetés contenant un ensemble de litéraux, appelé contexte. Cette nouvelle représentation permet d'étendre quelques caractéristiques intéressantes de l'algorithmique 2-SAT. Parmi elles, la résolution classique est reformulée en utilisant la fermeture transitive d'un graphe. Les chemins entre les noeuds permetent de donner une façon originale pour calculer les conditions minimales sous lesquels un litéral est impliqué.Deux utilisations concrètes de ce graphe sont présentés. La première concerne l'extraction des ensembles 2-SAT strong backdoor et la seconde consiste en une technique de pré-traitement des formules CNF. Des résultats expérimentaux prometteurs sont obtenus sur plusieurs classes d'instances issues des dernières compétitions.

1 Introduction

Le problème SAT, qui consiste à vérifier si un ensemble de clauses est satisfaisable ou pas, est central en informatique et dans le domaine de l'intelligence artificielle. Il est notamment utilisé pour la preuve de théorème, la planification, le raisonnement non monotone, la vérification de circuits. Durant ces deux dernières décennies, plusieurs approches ont été proposées pour résoudre des instances difficiles en utilisant des algorithmes complets ou incomplets. La recherche locale (e.g. [24]), les variantes élaborées à partir de la procédure DPLL de Davis-Putnam-Loveland-Logemann [7] (e.g. [22, 12]) permettent maintenant de résoudre des instances de plus en plus grandes issues du monde réel. La plupart des solveurs complets sont basés sur la technique du backtrack initiée par Davis Putnam Logemann Loveland (DPLL). Ces algorithmes de base intègrent des techniques de plus en plus efficaces comme l'apprentissage, la propagation de contraintes, le pré-traitement, l'exploitation des symétries, etc. L'impact de ces différentes améliorations dépend du type des instances à résoudre. Par exemple, l'apprentissage est plus efficace sur les instances issues du monde réel que sur les instances aléatoires. Un autre aspect important lié à l'efficacité des solveurs SAT concerne le codage des problèmes traditionnellement transformées sous forme normale conjonctive (CNF). Cependant, le codage sous forme CNF induit une perte des connaissances

structurelles qui sont mieux exprimées dans d'autres formalismes et qui peuvent permettre une résolution plus efficace [16, 14, 26]. Pour exploiter ces connaissances structurelles, des travaux récents ont été proposés. Quelques uns utilisent la forme étendue des formules booléennes (nonCNF [26], contraintes pseudo booléennes [9]) pour le codage des formules. Tandis que d'autres essayent de récupérer et/ou déduire des propriétés structurelles à partir des formules CNF (symétries [1], dépendances fonctionnelles [14], équivalences [17]). Finalement, différents graphes ont été élaborés pour présenter ou modéliser ou bien même résoudre des instances SAT. Ce dernier type d'approche est motivé par la visualisation de structures [25], la décomposition [6], la propagation [18], l'apprentissage (à partir du graphe d'implication) [21, 22].

Dans cet article, un nouvelle représentation en graphe est proposée. Elle étend de manière originale le graphe d'implication pour les formules binaires (2-SAT) au cas général. Chaque clause est représentée par un ensemble d'implications (conditionnelles) et codée avec différents arcs étiquetés avec un ensemble de litéraux, appelé contexte (ou condition). Cette nouvelle représentation permet d'étendre quelques caractéristiques intéressantes de l'algorithmique 2-SAT. Parmi elles, la résolution classique est reformulée en utilisant la fermeture transitive du graphe. Les chemins entre les noeuds opposés permettent d'avoir des conditions pour la définition du backbone i.e. les conditions minimales sous lesquelles un litéral est impliqué. Cependant, dans le cas général, trouver l'ensemble minimal sous lequel un litéral est impliqué est intraitable. Clairement, cette nouvelle représentation admet plusieurs utilisations intéressantes.

Dans cet article, deux manières d'exploiter cette représentation sont décrites. La première concerne le calcul des ensembles strong backdoor et la seconde est une technique de pré-traitement des formules CNF. Le reste de l'article est organisé comme suit. Après quelques définitions préliminaires, le graphe associé à une formule CNF est décrit. Une description formelle de ces caractéristiques est détaillée. Plus particulièrement la résolution est reformulée en utilisant la fermeture transitive du graphe. Deux utilisations pratiques de cette représentation sont ensuite proposées. Nous terminons par une étude expérimentale menée sur des instances issues des dernières compétitions SAT.

2 Préliminaires

Soit $\mathcal{B}$ un langage booléen (i.e. propositionnel) de formules construites classiquement, en utilisant les connecteurs usuels ($\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftrightarrow$) et un ensemble de variables propositionnelles. Une *formule CNF* Σ est un ensemble (interprété comme une conjonction) de *clauses*, et une clause est un ensemble (interprété sous forme disjonctive) de *litéraux*. Un litéral est une variable ou sa négation. Pour un litéral x (resp. une clause c), $\mathcal{V}(x)$ (resp. $\mathcal{V}(c)$) correspond à la variable propositionnelle associé au litéral x (resp. l'ensemble de variables associées à c). On note $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) l'ensemble des variables (resp. litéraux) apparaissant dans Σ. La taille d'une clause c, noté $|c|$, est égale au nombre de ces litéraux . Une *clause unitaire* est une clause de taille un. Un *litéral unitaire* est l'unique litéral d'une clause unitaire. Une clause est dite *clause binaire* si elle contient exactement deux litéraux. Une *clause* est *Horn* si elle contient au plus un litéral positif. On appelle clause *positive* (resp. *négative*) toute clause contenant uniquement des litéraux positifs (resp. négatifs). Une formule est *Horn-SAT* (rep. *2-SAT*) si elle est

composée uniquement de clauses de Horn (clauses binaires). Ces deux fragments SAT sont connus pour être traitables en temps linéaire [2, 10].

En plus de ces notations usuelles, on définit la négation d'un ensemble de litéraux S comme l'ensemble des opposés de chaque litéral de S, e.g. $\neg S = \{\neg l | l \in S\}$.

Une *interprétation* I d'une formule booléenne est une affectation des valeurs de vérité de ces variables. I peut être représentée sous forme d'une conjonction (un ensemble) de litéraux. Soit $l \in \mathcal{L}(\phi)$, la formule obtenue en affectant l à vrai, c'est à dire $\phi(l) = \{c - \{\neg l\} | c \in \phi, \neg l \in c\} \cup \{c | c \in \phi, l \notin c, \neg l \notin c\}$. Pour une interprétation $I = \{l_1, l_2, \ldots, l_n\}$, $\phi(I) = \phi(l_1)(l_2)\ldots(l_n)$. Un *modèle* d'une formule est une interprétation I qui satisfait cette formule i.e. $\phi(I) = \emptyset$. En conséquence, résoudre le problème SAT consiste à trouver un modèle d'une formule CNF si ce modèle existe ou prouver l'inexistence d'aucun modèle pour cette formule.

Soit c_1 une clause contenant un litéral a et c_2 une clause contenant le litéral opposé $\neg a$, la *résolvante* entre c_1 et c_2 est la disjonction de tous les litéraux de c_1 et c_2 excepté a et $\neg a$, notée $res(a, c_1, c_2)$. Une résolvante est dite *tautologique* si elle contient un litéral et son opposé, sinon elle est dite fondamentale.

3 Graphe associé à une formule CNF

Comme mentionné dans l'introduction, différentes représentations sous forme de graphe d'une formule CNF ont été proposées. Elles ont induites différentes utilisations, comme le graphe d'apprentissage [22], le pré-traitement [4], la survey propagation pour résoudre les instances aléatoires 3-SAT [5], la visualisation [25]. Notre représentation peut être vue comme une extension du graphe d'implication 2-SAT [2] au cas général.

3.1 Vers un graphe étiqueté

Avant d'introduire notre approche, rappelons le graphe d'implication des formules 2-SAT.

Soit ϕ une formule 2-SAT. Le graphe associé à ϕ, est définit comme $G_\phi = (S, E)$, où $S = \{x, \neg x | x \in \mathcal{L}(\phi)\}$ et $E = \{(\neg x, y), (\neg y, x) | (x \vee y) \in \phi\}$. L'algorithme polynômial utilisé pour résoudre ϕ est basé sur l'application de la fermeture transitive de G_ϕ, $G_\phi^c = (S, E')$ et sur la vérification qu'il existe un litéral $x \in S$ tel que $(x, \neg x) \in E'$ et $(\neg x, x) \in E'$.

Évidemment, le calcul de la fermeture transitive du graphe G_ϕ est équivalent à la saturation de ϕ par résolution. En effet, pour (x, y) et (y, z) de E un nouveau arc (x, z) est généré et ajouté au graphe. Une telle application correspond à $res(y, (\neg x \vee y), (\neg y \vee z))) = (\neg x \vee z)$.

Pour une formule CNF quelconque, une représentation naturelle peut être obtenue en utilisant un hypergraphe où les noeuds sont représentés par les litéraux, et les hyper arrêtes correspondent aux clauses.

Dans ce qui suit, le graphe représentatif d'une formule CNF générale est décrit.

Définition 1
Soit c une clause telle que $|c| \geq 2$. Un contexte η_c associée à c est une conjonction de

litéraux tel que $\neg\eta_c \subset c$ *et* $|c - \{\neg\eta_c\}| = 2$ *i.e quand* η_c *est vrai, la clause* c *devient une clause binaire.*

Exemple 1
Soit $c = (a \vee \neg b \vee c \vee d)$ *une clause. Un contexte possible associé à* c *est* $\eta_c = (b \wedge \neg c)$. *La clause* c *peut être réécrite comme* $((\neg a \wedge \eta_c) \rightarrow d)$.

Pour une clause c de taille k, on a $|\eta_c| = k - 2$. Le contexte associé à une clause binaire est vide, Tandis que pour les clauses ternaires, le contexte est réduit à un seul litéral. Le nombre des contextes possibles de c est égal à $\frac{k(k-1)}{2}$. Une clause c peut être réécrite en $k(k-1)$ différentes façons. Quand $k = 1$, la clause est unitaire, aucun contexte n'est possible.

En utilisant la clause c de l'exemple 1, on obtient six contextes possibles de taille 2 et douze différentes façons pour réécrire c.
Définissons maintenant le graphe représentatif d'une formule CNF.

Définition 2 (Représentation SAT-graphe)
Soit ϕ *une formule CNF. On définit* $G_\phi = (S, E, v)$ *comme étant le graphe SAT associée à* ϕ *défini comme suit*
- $S = \{x, \neg x | x \in \mathcal{L}(\phi)\}$
- $E = \{a = (\neg x, y)$ *tel que* $\exists c \in \phi, c = (x \vee \neg\eta_c \vee y)$ *et* $v(a) = \eta_c\}$

Dans la suite, pour des raisons de clarté, on notera parfois l'arc étiqueté $a = (x, y)$ comme $(x, y, v(a))$. On note aussi $cla(a) = (\neg x \vee \neg v(a) \vee y)$ comme étant la clause associée à a.

Clairement la définition 2 généralise le graphe classique 2-SAT. En effet, si tous les contextes sont vide i.e. toutes les clauses de ϕ sont binaires, dans ce cas tous les arcs de G_ϕ sont étiquetés avec un ensemble vide de litéraux.

3.2 Résolution/fermeture transitive

Dans cette section, nous démontrons l'équivalence entre la résolution classique et la fermeture transitive du graphe. On commencera par introduite quelques définitions nécessaires et on considère dorénavant uniquement les formules ϕ sans clauses tautologiques et $G_\phi = (S, A, v)$ indique le graphe associé à ϕ.

Définition 3
Soit $G_\phi = (S, A, v)$ *un graphe,* $a_1 = (x, y, v(a_1)) \in A$ *et* $a_2 = (y, z, v(a_2)) \in A$. *On définit* $tr(a_1, a_2) = a_3$ *tel que* $a_3 = (x, z, v(a_1) \cup v(a_2) \backslash \{x, \neg z\})$.

Dans la définition ci-dessus, l'élimination de $\{x, \neg z\}$ du contexte associé à a_3 garantie que la clause $cla(a_3)$ ne contient pas plusieurs occurrences du même litéral. Ce qui correspond à l'application de la règle de fusion.

Exemple 2
Soit $a_1 = (x, y, \{\neg z, e\})$ *et* $a_2 = (y, z, \{x, f\})$. *Les deux clauses codées respectivement par* a_1 *et* a_2 *sont* $c_1 = (\neg x \vee z \vee \neg e \vee y)$ *et* $c_2 = (\neg y \vee \neg x \vee \neg f \vee z)$. *On obtient*

$tr(a_1, a_2) = (x, z, \{\neg e, \neg f\})$ *et* $cla(tr(a_1, a_2)) = (\neg x \vee \neg e \vee \neg f \vee z)$*. Cette dernière clause ne contient pas de litéraux redondants. En utilisant la résolution,* $res(c_1, c_2, y)$ *on obtient donc* $c_3 = (\neg x \vee z \vee \neg e \vee \neg x \vee \neg f \vee z)$*.*

En appliquant la règle de fusion sur c_3*, on élimine une occurrence de* $\neg x$ *et* z*. On obtient* $res(c_1, c_2, y) = cla(tr(a_1, a_2)) = (\neg x \vee \neg e \vee \neg f \vee z)$

Propriété 1
Soit $c_1 = (x \vee \eta_{c_1} \vee y) \in \phi$*,* $c_2 = (\neg y \vee \eta_{c_2} \vee z) \in \phi$*,* $a_1 = (x, y, \eta_{c_1}) \in A$ *et* $a_2 = (x, y, \eta_{c_2}) \in A$*. On a* $res(c_1, c_2, y) = cla(tr(a_1, a_2))$

La preuve de la propriété 1 est une conséquence directe des définitions 2 et 3.

Définition 4 (chemin)
Un chemin $p(x, y)$ *entre* x *et* y *dans* G_ϕ*, est définit comme suit :* $p(x, y) = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$ *tel que* $x_{i_1} = x$ *et* $x_{i_k} = y$ *et* $1 < j \leq k$ $(x_{i_{j-1}}, x_{i_j}) \in A$*.*
On définit $\eta_p = \bigcup_{1<j\leq k} v((x_{i_{j-1}}, x_{i_j}))$ *le contexte associée à* $p(x, y)$
et $tr(p(x, y)) = tr(\ldots(tr((x_{i_1}, x_{i_2}), (x_{i_2}, x_{i_3}))\ldots(x_{i_{k-1}}, x_{i_k})\ldots)$*,*
comme étant la fermeture transitive associée à $p(x, y)$*.*

Définition 5 (chemin fondamental)
Soit $p(x, y) = [x = x_{i_1}, x_{i_2}, \ldots, x_{i_k} = y]$ *un chemin entre* x *et* y*.* $p(x, y)$ *est dit fondamental s'il satisfait les conditions suivantes :*
- η_p *ne contient pas un litéral et son opposé.*
- $\neg x \notin \eta_p$ *et* $y \notin \eta_p$

La propriété suivante montre qu'à partir d'un chemin fondamental $p(x, y)$ on peut dériver une résolvante fondamentale en utilisant la fermeture transitive.

Propriété 2
Soit $p(x, y)$ *un chemin.* $p(x, y)$ *est fondamental ssi* $cla(tr(p(x, y)))$ *est une clause fondamentale.*

Preuve 1
Pour un chemin fondamental $p(x, y)$*,* η_p *ne contient pas deux litéraux opposés. Comme* $cla(tr(p(x, y))$ *peut être réécrite sous forme* $r = (\neg x \vee \neg \eta_p \vee y)$*,* r *est clairement une clause fondamentale. En effet, d'après la définition 5,* η_p *ne contient pas un litéral et son opposé (première condition) et* $x \in \neg\eta_p$ *et* $\neg y \notin \neg\eta_p$ *(seconde condition). L'inverse est aussi vrai. On suppose que* $cla(tr(p(x, y)))$ *n'est pas fondamentale. Comme* $tr(p(x, y)) = (x, y, \eta_p)$*, la clause* $cla(tr(p(x, y)) = (\neg x \vee \neg \eta_p \vee y)$ *n'est pas fondamentale. Ce qui montre que la première ou la deuxième condition de la définition 5 n'est pas satisfaite.*

Dans ce qui suit, on décrit comment la résolution classique peut être appliquée en utilisant la fermeture transitive.

Définition 6
Soit $G_\phi = (S, A, v)$ *le graphe associé à* ϕ*. On définit* $tr(G_\phi) = (S, A', v')$ *tel que :* $\forall x \in S$*,* $\forall y \in S$ *tel que* $\exists p(x, y)$ *un chemin fondamental entre* x *et* y*,* $A' = A \cup \{tr(p(x, y))\}$*.*

Propriété 3
Soit ϕ une formule CNF. ϕ est insatisfiable ssi $\exists k$ tel que $tr^k(G_\phi) = (S, A', v')$ et $\exists x \in S$ tel que $(x, \neg x, \emptyset) \in A'$ et $(\neg x, x, \emptyset) \in A'$

Preuve 2
Notre graphe est complet i.e. chaque clause est codée $k(k-1)$ fois. Comme montré précédemment, l'application de tr sur les arcs de G_ϕ est équivalent à l'application de la résolution entre les clauses de ϕ.

La preuve pour la réfutation complète peut être dérivée de la résolution classique.

Dans la propriété 3, nous avons montré comment on peut appliquer la résolution classique sur le graphe. Évidemment, d'autres techniques SAT (e.g. élimination de variable, propagation unitaire) peuvent être reformulées en utilisant ce même graphe. Notons que ce graphe peut avoir plusieurs caractéristiques intéressantes. L'un des principaux avantages est que la dépendance entre les clauses est mieux exprimée en utilisant une telle représentation. Une telle propriété structurelle est connue pour être déterminante dans l'efficacité des solveurs SAT. Plus intéressant encore, le graphe SAT peut être vu comme un graphe d'états, où chaque état représente un litéral et une transition entre les deux états s_1 et s_2 est représentée par une condition $v((s_1, s_2))$ (un ensemble de litéraux). Plusieurs problèmes intéressants peuvent être reformulés plus facilement. Par exemple, le calcul du plus court chemin entre un litéral et son opposé i.e. le calcul des conditions minimales pour qu'un litéral puisse être impliqué.

Dans la section suivante, deux exploitations possibles de notre graphe sont décrites.

4 Utilisation de representation sous forme de graphe

Dans la définition de G_ϕ, chaque clause est représenté $k(k-1)$ fois. Une telle représentation multiple d'une clause donnée est nécessaire pour que la fermeture transitive soit complète pour la réfutation. Pour des formules de grandes tailles, la représentation complète du graphe est impraticable.

Suivant à quelle fin ce graphe est utilisé, des restrictions utiles peuvent être définies. Par exemple, si on représente chaque clause avec seulement un seul arc, le graphe obtenu est équivalent à la formule CNF.

Nous devons seulement nous rappeler que dans ce cas certaines résolvantes pouvant être obtenues en employant la fermeture transitive du graphe total ne peuvent plus être dérivées à partir de ce graphe restreint.

4.1 Ensembles 2-SAT strong backdoor

Dans cette section, on va montrer comment la représentation sous forme de graphe peut être utilisée pour le calcul des ensembles strong backdoor.

La notion de (strong) backdoor introduite par Williams-etal in [27] est un domaine de recherche très actif actuellement. Un ensemble de variables forme un backdoor d'une formule, si il existe une affectation de ces variables rendant la formule simplifiée traitable en temps polynômial. Un tel ensemble de variables est strong backdoor si pour

toute affectation de ces variables le reste de sous-formule est traitable polynomialement. Ce type de structure est relié à la notion de variables indépendantes [15, 13] et à l'ensemble coupe cycle introduit pour le problème de la satisfaction de contraintes [8].

Rappelons que le calcul de l'ensemble strong backdoor minimal est un problème NP-difficile. En pratique, approximer un strong backdoor de taille "raisonnable"est un challenge très intéressant.

Des précédents travaux ont été effectués dans ce but. Par exemple, l'approche proposée dans [14] essaye de couvrir un ensemble de portes (fonctions booléennes) à partir d'une formule donnée, et en exploitant les dépendances entre les variables un strong backdoor est calculé.

Le principal inconvénient de cette approche est que plusieurs problèmes ne contiennent pas de telles propriétés structurelles (fonctions booléennes). Recemment, dans [20] un concept de fraction reverse horn d'une formule CNF est proposé et une importante corrélation entre la densité des instances aléatoires 3-SAT et les performances des solveurs SAT est mise en évidence. Dans [19], la relation entre la complexité des instances aléatoires 3-SAT insatisfaisables et les noyaux et les ensembles strong backdoor est étudiée.

Comme notre représentation sous forme de graphe est une généralisation du graphe d'implication 2-SAT, les ensembles strong backdoor que nous allons calculer sont considérés par rapport au fragment polynômial 2-SAT i.e. l'ensemble de variables B tel que pour toute affectation I des variables de B, la formule simplifiée par I appartient à la classe 2-SAT.
Notons que pour calculer les ensembles 2-SAT strong backdoor, il est suffisant de considérer une restriction du graphe. Chaque clause de ϕ est représentée avec seulement un seul arc dans G_ϕ. On note G^u_ϕ le graphe restreint associé à ϕ, appelé graphe unique.
La propriété suivante montre comment on peut utiliser le graphe unique G^u_ϕ pour calculer les ensembles 2-SAT strong backdoor.

Propriété 4
Soit ϕ une formule, et $G^u_\phi = (S, A, v)$ son graphe associé.
$B = \cup_{a_i \in A}\{\mathcal{V}(l) | l \in v(a_i)\}$ est un ensemble 2-SAT strong backdoor.

Preuve 3
Pour prouver cette propriété, il est suffisant de montrer que pour toute affectation I de B, $\phi(I)$ est une formule 2-SAT. Chaque arc $a_i = (x, y, v(a_i)) \in A$ code une clause $c = (\neg x \vee \neg v(a_i) \vee y) \in \phi$. Pour chaque clause $c \in \phi$, toutes les variables de $v(a_i)$ apparaissent dans B. Conséquence I soit il satisfait au moins un litéral dans $\neg v(a_i)$, soit tous les litéraux dans $\neg v(a_i)$ sont faux. En Conséquence, dans tous les cas, la formule est soit 2-SAT, non satisfaite ou satisfaite. B est donc un 2-SAT strong backdoor.

Rappelons que notre objectif principal est l'approximation de l'ensemble strong backdoor 2-SAT minimal. Pour arriver à cette fin, on utilise G^u_ϕ le graphe unique de ϕ. De plus, comme une clause c de taille k peut être codée de plusieurs façons, nous avons besoin de choisir un arc parmi les $k(k-1)$ possibles. Pour construire G^u_ϕ, à chaque étape une nouvelle clause $c = (x \vee \eta_c \vee y) \in \phi$ de taille k est traitée et un nouvel arc $a = (x, y, v(a))$ où $v(a) = \neg\eta_c$, est ajouté au graphe. Premièrement, tous les litéraux

de c appartenant aux précédents contextes sont inclus dans $v(a)$. Deuxièmement, les littéraux x et y sont choisi parmi les littéraux restants (ceux qui apparaissent le moins dans ϕ). Plus intéressant encore, la propriété suivante est utilisé pour réduire encore plus la taille de l'ensemble strong backdoor.

Propriété 5
Soit B un ensemble 2-SAT strong backdoor d'une formule CNF ϕ, et $b \in B$. si $\forall c \in \phi$ tel que $b \in c$ or $\neg b \in c$, $|\mathcal{V}(c) \cap B| > |c| - 2$, alors la variable b peut être retirée de B.

La preuve de la propriété 5 est évidente. Cette dernière exprime le fait qu'une variable de l'ensemble 2-SAT strong backdoor peut être éliminée si elle apparaît seulement dans les clauses avec au plus un litéral n'appartenant pas à B i.e. l'élimination de telles variables aboutit à des clauses contenant au plus deux litéraux qui ne sont pas dans B.

L'algorithme 1décrit notre approche pour le calcul du strong backdoor 2-SAT. Comme nous cherchons des ensembles 2-SAT strong backdoor, toutes les clauses binaires de ϕ sont supprimées (ligne 2), et en utilisant l'heuristique décrite précédemment, le graphe G^u_ϕ est construit sur l'ensemble des clauses restantes. Un premier ensemble B est calculé (ligne 4), et réduit en utilisant la propriété 5 (ligne 5 à 10).

Input: une formule CNF ϕ
Output: un ensemble 2-SAT strong backdoor B
1 **begin**
2 Supprimer les clauses binaires de ϕ
3 Créer le graphe $G^u_\phi = (S, E)$
4 $B = \bigcup_{a \in E} \mathcal{V}(v(a))$
5 **foreach** $c \in \phi$ **do** $nb(c) = |\{x \notin c \cap B\}|$
6 **foreach** $u \in B$ **do**
7 bool remove=true
8 **foreach** $(c \in \phi | u \in c$ *or* $\neg u \in c)$ **do**
9 **if** $(nb(c) = 2)$ **then** remove = false
10 **end**
11 **if** *remove* **then** $B = B - \{u\}$
12 **end**
13 **end**

Algorithm 1: Approximation d'un ensemble 2-SAT strong backdoor

4.2 Graphe et pré-traitement

La simplification des formules CNF est un moyen très important de pré-traitement inclus dans de nombreux solveurs [22]. Plusieurs techniques de pré-traitement ont été proposées. Citons les plus récentes comme "Sattelite" qui utilise la technique de l'élimination de variables [11] et "hypre" un pré-traitement basé sur l'hyper binaire résolution [3].

Dans cette section, un nouveau pré-traitement basé sur la représentation graphique d'une formule CNF est proposé, on y considère le graphe restreint suivant.

Définition 7 (Graphe SAT ordonné)
Soit ϕ une formule. On définit le graphe ordonné G^{or}_ϕ comme suit :

- $S = \{x, \neg x | x \in \mathcal{L}(\phi)\}$
- $E = \bigcup_{c \in \phi} arcs(c)$, *tel que* $arcs(c)$ *pour* $c = (x_1 \vee x_2 \vee .. \vee x_n)$ *est défini comme :*
  - *pour* $(1 \leq i < n)$, $a_i = (x_i, x_{i+1}, v(a_i))$ *et* $v(a_i) = \{x_j | 1 \leq j \leq n$ *et* $j \neq i$ *et* $j \neq i + 1\}$, *et*
  - *pour* $(i = n)$, $a_n = (x_n, x_1, v(a_n))$ *et* $v(a_n) = \{x_2, ..., x_{n-1}\}$

Exemple 3
Soit $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x) \wedge (\neg x_2 \vee x) \wedge (\neg x_3 \vee x)$ *une formule CNF. Le graphe ordonné* G^{or}_{ϕ} *est représenté dans la figure 1. Comme on peut le remarquer, pour les clauses binaires, le graphe obtenu (SAT graphe ordonné) est le même avec et sans resctriction. i.e. une clause binaire est representé par deux arcs dans les deux graphes.*

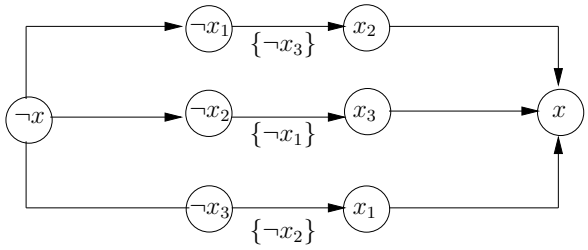

FIG. 1 – Graphe G^{or}_{ϕ} ordonné associé à la formule ϕ de (exemple 3)

Le graphe ordonné est un bon compromis entre une représentation totale et le graphe unique. Comme expliqué dans les sections précédentes, l'utilisation de la fermeture transitive sur un chemin du graphe, permet de générer facilement des résolvantes. L'application de la résolution classique revient à appliquer la fermeture transitive du graphe. En conséquence, un tel processus est clairement exponentiel dans le pire des cas.

Décrivons maintenant l'étape principale de notre pré-traitement utilisant le graphe représentatif d'une formule CNF ϕ. Le but de cette étape est de générer une résolvante fondamentale de taille limitée. A chaque étape une nouvelle clause $c = (x_1 \vee x_2 \vee \ldots \vee x_n)$ de ϕ est sélectionnée et traitée comme décrit dans l'algorithme 2. Pour chaque $x_i \in c$ (ligne 4), un arc $a = (x_i, x_j, v(a)) \in G^{or}_{\phi}$ est choisi tel que $res(c, (\neg x_i \vee \neg v(a) \vee x_j), x_i)$ est fondamentale (ligne 4 à 11). Plus précisément, la nouvelle résolvante est générée suivant les arcs de G^{or}_{ϕ}. Une telle résolvante est composée des ensembles de litéraux cumulées ($noeuds$) et contextes (η). Un arc a est choisi tel que $res(c, cla(a), x_i)$ est fondamentale (ligne 5). Si un tel arc existe, le contexte η (resp. $noeuds$) est augmenté de $a(v)$ (resp. $\{x_j\}$) (ligne 6) ; sinon x_i est simplement ajouté à l'ensemble des noeuds (ligne 8) si la résolvante obtenue est fondamentale. (ligne 7). Dans le dernier cas, si aucune clause fondamentale n'a pu être générée le pré-traitement s'arrête (ligne 10).

Pour une formule contenant une clause $c = (x_1 \vee x_2 \vee x_3)$ et un ensemble de clauses binaires $C = \{(\neg x_1 \vee y), (\neg x_2 \vee y), (\neg x_3 \vee y)\}$, le litéral y est généré par l'algorithme 2. A partir de la clause c et les arcs qui codent C. Dans ce cas, l'application de notre algorithme correspond exactement à l'application de l'hyper binaire

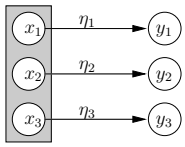

Fig. 2.a résolution par le graphe

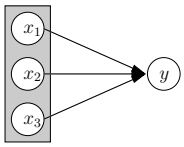

Fig. 2.b Hyper binaire résolution

résolution sur c et C [3] (see figure 2.b). Pour les clauses C de taille quelconque ($C = \{(\neg x_1 \vee \neg\eta_1 \vee y_1), (\neg x_2 \vee \neg\eta_2 \vee y_2), (\neg x_3 \vee \neg\eta_3 \vee y_3), \}$) (voir figure 2.a), notre algorithme génère la résolvante $(\neg\eta_1 \vee \neg\eta_2 \vee \neg\eta_3 \vee y_1 \vee y_2 \vee y_3)$. Clairement, une étape de notre pré-traitement comme décrit au-dessus peut être vue comme une extension de l'hyper binaire résolution proposée par [3].

Notre pré-traitement utilise deux paramètres fixes, le nombre de clauses considérées ($nbCla$) et la taille maximale de la résolvante générée ($maxRes$). Pour chaque clause c l'algorithme 2 est itéré. Ce processus est réitéré en utilisant la résolvante obtenue dans l'étape précédente, jusqu'à ce qu'aucune résolvante fondamentale ne puisse être obtenue.

Input: $G^{or}(V, E)$ la représentation en graphe de ϕ
$c \in \phi$
Output: c_{res} une clause résolvante
begin
  $\eta = \emptyset$
  $noeuds = \emptyset$
  foreach $x_i \in c$ **do**
    if $\exists a = (x_i, x_j, v(a)) \in E$ *tel que*
    $x_j \notin \eta$ *et* $v(a) \cap noeuds = \emptyset$ *et* $\neg x_j \notin c$ *et* $\neg v(a) \cap c = \emptyset$
    then $\eta = \eta \cup v(a)$, $noeuds = noeuds \cup \{x_j\}$
    else if *(*$x_i \notin \eta$ *et* $\neg x_i \notin noeuds$*)*
    then $noeuds = noeuds \cup \{x_i\}$
    else
    return *null*
    $c_{res} = noeuds \cup \neg\eta$
  end
  return c_{res}
end

Algorithm 2: Une étape du pré-traitement

5 Expérimentations

Les résultats expérimentaux reportés dans cette section ont été conduits sur de nombreuses instances SAT issues des dernières compétitions et réalisés sur un Xeon 3.2 GHz (2 GB RAM).

La table 1 présente une comparaison entre notre approche pour calculer l'ensemble 2-SAT strong backdoor (algorithme 1) et celle calculant l'ensemble Horn strong backdoor proposée récemment dans [23]. Pour chaque instance, le nombre de variables (#V), le

instance	#V	#C	Horn BD	2SAT BD
clauses-4-1967	267767	823658	–	60747 (22%)
clauses-2-1966	75528	226124	29357 (38%)	17945 (23%)
depots1_ks99i-4010	161	445	32 (19%)	31 (19%)
equilarge_l2-519	4487	18555	2582 (57%)	367 (8%)
equilarge_l3-520	4496	18591	2595 (57%)	373 (8%)
equilarge_l1-518	4574	18903	2635 (57%)	374 (8%)
ferry6_ks99i.renamed-sat05-3997	1425	14454	657 (46%)	539 (35%)
ferry8_ks99a.renamed-sat05-4004	1259	15206	573 (45%)	471 (37%)
linvrinv7-566	1274	4166	631 (49%)	467 (36%)
linvrinv8-567	1920	6337	961 (50%)	705 (36%)
linvrinv5-564	450	1426	221 (49%)	167 (37%)
iso-brn100-3025	3587	5279	372 (10 %)	349 (9%)
iso-brn009-2934	1770	4540	318 (17 %)	187 (10%)
iso-icl009-3243	2251	5177	442 (19 %)	512 (22%)
iso-ukn006-3387	1906	4188	407 (21%)	453 (23%)
iso-icl008-3242	2136	5938	461 (21%)	545 (25%)
mm-2x3-8-8-s.1-476	1148	8088	621 (54%)	132 (11%)
mm-2x3-8-8-sb.1-475	2326	80891	1661 (71%)	926 (39%)
mod2-rand3bip-unsat-120-1-2624	120	320	85 (69%)	50 (41%)
mod2c-rand3bip-unsat-120-3-2340	160	640	126 (78%)	86 (53%)
mod2c-rand3bip-unsat-150-1-2368	200	800	154 (77%)	111 (55%)
mod2c-rand3bip-unsat-105-2-2324	140	560	109 (77%)	78 (55%)
mod2c-rand3bip-unsat-120-1-2338	160	640	126 (78%)	90 (56%)
mod2-rand3bip-unsat-150-2-2655	150	400	106 (70%)	61 (40%)
par32-3-c	1325	5294	698 (52%)	199 (15%)
par32-1-c	1315	5254	696 (52%)	202 (15%)
par32-4-c	1333	5326	703 (52%)	202 (15%)
par32-5-c	1339	5350	708 (52%)	202 (15%)
par16-5-c	341	1360	189 (55%)	70 (20%)
par16-1-c	317	1264	176 (55%)	68 (21%)
pmg-12-UNSAT-3940	190	632	136 (71%)	77 (40%)
pmg-14-UNSAT-3942	577	1922	408 (70%)	232 (40%)
pmg-11-UNSAT-3939	169	562	122 (72%)	65 (38%)
pmg-13-UNSAT-3941	409	1362	291 (71%)	163 (39%)
series18	6410	32421	1665 (25%)	1122 (17%)
series16	4546	22546	1189 (26%)	870 (19%)
strips-gripper-08t14-1156	1966	23326	826 (42%)	773 (39%)
strips-gripper-16t31-1146	8904	222172	4176 (46%)	3788 (42%)
strips-gripper-18t35-1147	11318	316416	5331 (47%)	4835 (42%)
strips-gripper-10t19-1143	3390	53008	1475 (43%)	1400 (41%)
satellite3_v01a-3989	303	7840	226 (74%)	210 (69%)

TAB. 1 – 2-SAT vs Horn strong backdoor

instance	SAT	#V	#C	#CA	preproc	Minisat +preproc	Minisat
mod2-r3b.280-1-2263	Y	280	1120	183	0.76	574.49	–
mod2-r3b.240-1-2203	Y	240	960	162	0.73	250.35	–
mod2-r3b.270-1-2248	Y	270	1080	180	0.76	510.86	–
mod2-r3b.260-2-2234	Y	260	1040	190	0.74	657.93	–
mod2c-r3b.210-2-2474	Y	297	2092	341	1.14	488.36	–
mod2c-r3b.180-2.05-2429	Y	252	1784	303	1.08	204.37	411.16
mod2c-r3b.170-3.05-2415	Y	240	1724	276	1.04	286.37	368.76
mod2-r3b.150-1.05-2654	N	150	400	79	0.52	160.75	189.16
mod2-r3b.135-2.05-2640	N	135	360	71	0.50	336.46	828.83
mod2-r3b.135-1.05-2639	N	135	360	70	0.51	529.64	603.32
mod2-r3b.250-1.05-2218	Y	250	1000	169	0.72	699.16	753.9
fclqcolor-08-05-06-1273	N	116	1362	96	0.77	36.84	66.88
fphp-012-011.05-1228	N	132	1398	15	0.78	889.1	651.32
gensys-icl009-3135	N	926	17577	1375	6.63	759.11	–
gensys-icl008-3134	N	960	17640	1482	6.74	889.67	–
gensys-icl006.05-2718	N	1321	7617	2360	70.78	189.45	203.492
gensys-ukn001.05-2678	N	1886	7761	2260	70.57	655.86	728.51
gensys-ukn006.05-3349	N	874	16339	1203	5.56	410.73	365.82
gt-020-1305	N	400	7050	770	2.33	821.44	–
gt-018-1292	N	324	5066	577	1.81	18.79	127.74
grid-pbl-1342.05-1342	N	3660	7142	1483	3.98	350.32	192.17
grid-pbl-1342.05-1342	N	3660	7142	1489	7.06	64.31	153.51
homer11.shuffled	N	220	1122	28	0.71	48.94	59.99
homer09.shuffled	N	270	1920	1920	5.18	57.23	70.32
lisa21_0_a	Y	1453	7967	1401	2.92	32.67	77.04
php-012-012-1158	Y	144	804	23	0.63	66.3	210.19
strips-gripper-14t27-1145	Y	6778	93221	53	48.0	199.36	–
strips-gripper-12t23-1144	Y	4940	148817	38	26.49	167.44	–
vmpc_34-1958	Y	1156	194072	21109	53.63	60.86	–
vmpc_26-1946	Y	676	86424	9612	22.68	209.26	306.37
vmpc_27-1947	Y	729	96 849	10729	24.63	596.71	254.15
clqcolor-08-05-0605-1249	N	116	1114	94	4.8	48.39	85.80
series16	Y	4546	22546	4298	10.43	51.62	316.14
clus-45-30-003.03-1082	Y	1200	4800	1004	5.82	85.73	197.12
clus-20-20-003.sat03-1087	Y	1208	4800	982	6.17	77.39	174.89

TAB. 2 – Minisat [12] avec et sans pré-traitement

nombre de clauses (#C) et la taille de l'ensemble 2-SAT (resp. Horn) strong backdoor est reporté. Ces résultats montrent clairement que notre algorithme est meilleur que celui de Paris et al.

Dans quelques cas , le gain est très important (e.g. `mm-*`, `equilarge-*`, `iso-* series-*`). Sur un large panel d'instances, l'algorithme 2-SAT strong backdoor est le meilleur.

Notre seconde proposition concerne le pré-traitement des formules CNF. (section 4.2). Comme mentionné, notre pré-traitement utilise deux paramètres fixes. Le premier, est le nombre de clauses à filtrer qui est fixé à 10% des clauses initiales et le second et la taille maximale des clauses résolvantes générées qui est lui fixé à 50.

La table 2 montre une comparaison expérimentale du solveur Minisat [12] avec et sans notre pré-traitement. Le temps de calcul est donné en secondes et limité à 900 secondes.

Pour chaque instances, on reporte le nombre de variables (#V), le nombre de clauses (#C), le nombre de clauses ajoutées à la formule originale (#CA), le temps dû au pré-traitement (preproc) et le temps total (pré-traitement + résolution) mis par Minisat. Ici aussi, les résultats sont prométeurs et montrent clairement l'intérêt de faire ce genre de pré-traitement. Par exemple, notre pré-traitement permet de résoudre 12 instances que Minisat classique. L'utilité de telles résolvantes est clairement mis en évidence (e.g. `strips-gripper-*`, `homer-*` and `series-*`). Bien evidemment, concernant les instances faciles, notre pré-traitement décroit les performances de Minisat.

6 Conclusion

Dans cet article nous avons présenté une nouvelle représentation de formules CNF sous forme de graphe. Elle étend le graphe d'implication 2-SAT. Cette nouvelle représentation offre des perspectives intéressantes. La structure de la formule (dépendances des variables) est clairement mieux exprimée. Nous avons montré que la résolution peut être appliquée à ce graphe en utilisant la notion de fermeture transitive du graphe. Deux façons d'exploiter ce graphe ont été présentées. Une technique d'extraction d'ensembles 2-SAT strong backdoor est proposée améliorant des résultats précédement obtenus.

Un pré-traitement des formules booléennes sous forme CNF est proposé étendant l'hyper binaire résolution. L'intégration de ce pré-traitement montre une amélioration des résultats du solveur Minisat sur une large collection d'instances.

Finalement, cette nouvelle représentation sous forme de graphe offre de perspectives intéressantes pour les recherches futures. Parmi elles, on envisage de calculer les conditions minimales sous lesquelles un litéral est impliqué. (le plus court chemin entre un litéral et son opposé).

Références

[1] F. Aloul, A. Ramani, I. Markov, and Karem A. Sakallah. Shatter : Efficient breaking for boolean satisfiability. In *proceedings of DAC*, pages 836–839. ACM Press, 2003.

[2] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3) :121–123, 1979.

[3] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *proceedings of SAT*, pages 341–355, 2003.

[4] R. Brafman. A simplifier for propositional formulas with many binary clauses. In *proceddings of IJCAI*, pages 515–522, 2001.

[5] A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation : An algorithm for satisfiability. *Random Struct. Algorithms*, 27(2) :201–226, 2005.

[6] A. Darwiche. New advances in compiling CNF to decomposable negational normal form. In *Proceedings of ECAI*, pages 328–332, 2004.

[7] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, July 1962.

[8] R. Dechter. "enhancement schemes for constraint processing : Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3) :273–312, 1990.

[9] H. Dixon and M. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *proceedings of AAAI*, pages 635–640, 2002.

[10] W. Dowling and J. Gallier. Linear-time algorithms for testing satisfiability of propositional horn formulae. *journal of logic programming*, 3 :267–284, 1984.

[11] N. Eén and A. Biere. Effective preprocessing in sat through variable and clause elimination. In *proceedings of SAT*, pages 61–75, 2005.

[12] N. Eén and N. Sörensson. An extensible sat-solver. In *proceedings of SAT*, pages 502–518, 2003.

[13] E. Giunchiglia, M. Maratea, and A. Tacchella. Dependent and independent variables in propositional satisfiability. In *proceedings of ECLAI*, pages 296–307, 2002.

[14] E. Gregoire, B. Mazure, R. Ostrowski, and L. Sais. Automatic extraction of functional dependencies. In *proceedings of SAT*, volume 3542 of *LNCS*, pages 122–132, 2005.

[15] H. Kautz, D. McAllester, and B. Selman. Exploiting variable dependency in local search. In *"Abstracts of the Poster Sessions of IJCAI-97"*, 1997.

[16] H. Kautz and D. McAllester B. Selman. Exploiting variable dependency in local search. In *Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97"*, Nagoya (Japan), 1997.

[17] C. Li. Equivalent literal propagation in the dll procedure. *Discrete Applied Mathematics*, 130(2) :251–276, 2003.

[18] F. Lu, L. Wang, K. Cheng, and R. Huang. A circuit sat solver with signal correlation guided learning. In *proceedings of DATE*, pages 892–897, 2003.

[19] I. Lynce and J. Marques-Silva. Hidden structure in unsatisfiable random 3-sat : an empirical study. In *proceedings of ICTAI*, pages 246–251, 2004.

[20] H. Van Maaren and L. Van Norden. Correlations between horn fractions, satisfiability and solver performance for fixed density random 3-cnf instances. *Annals of Mathematics and Artificial Intelligence*, 44 :157– 177, 2005.

[21] J. Marques-Silva and K. Sakallah. GRASP : A new search algorithm for satisfiability. In *Proceedings of CAD*, pages 220–227, 1996.

[22] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient sat solver. In *Proceedings of DAC*, 2001.

[23] L. Paris, R. Ostrowski, L. Saïs, and P. Siegel. Computing horn strong backdoor sets thanks to local search. In *proceedings of ICTAI*, pages 139–143, 2006.

[24] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI*, pages 440–446, 1994.

[25] C. Sinz and E. Dieringer. Dpvis - a tool to visualize the structure of sat instances. In *proceedings of SAT*, pages 257–268, 2005.

[26] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *proceedings of CP*, pages 663–678, 2004.

[27] R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *proceedings of IJCAI*, pages 1173–1178, 2003.