

Dépendances fonctionnelles booléennes : Détection et Exploitation

Éric Grégoire, Bertrand Mazure, Richard Ostrowski et Lakhdar Saïs

CRIL CNRS & IRCICA – Université d’Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France
{gregoire,mazure,ostrowski,sais}@cril.univ-artois.fr

Résumé Dans ce papier, nous proposons une nouvelle technique d’extraction de dépendances fonctionnelles dans les formules booléennes. Cette technique repose sur une utilisation originale de la propagation de contraintes. Celle-ci est utilisée pour déduire des fonctions booléennes à partir d’une formule mise sous forme normale conjonctive (CNF). Ces fonctions sont utilisées ensuite pour déterminer un sous-ensemble de variables « essentielles » permettant de capturer la combinatoire de la formule. La technique proposée dans cet article permet de détecter sur de nombreuses classes d’instances un nombre important de fonctions comparativement aux approches connues.

Mots clés : SAT, fonction booléenne, raisonnement propositionnel et recherche.

1 Introduction

Les progrès effectués ces dernières années dans la résolution pratique du problème SAT permettent d’envisager aujourd’hui la résolution d’instances issues d’applications réelles (voir par ex. [1,2,3]). Alors qu’il reste une forte compétition dans le but d’implémenter des solveurs efficaces pour résoudre les instances k -SAT aléatoires [4], il y a aussi un réel intérêt pour l’implémentation de systèmes performants capables de résoudre des instances difficiles de grandes tailles issues du monde réel. De nombreuses instances sont proposées et des compétitions internationales (e.g. la compétition DIMACS en 1993, la compétition Beijing en 1996, les compétitions SAT 2001-2004) sont régulièrement organisées autour de ces instances structurées.

Il reste que la représentation sous forme normale conjonctive (CNF) de problèmes issus d’applications réelles ne permet pas une représentation simple et compact des connaissances ; alors qu’à l’inverse la logique propositionnelle lorsqu’elle est utilisée dans sa totalité permet de représenter simplement une grande variété de connaissances tout en préservant la structure de la formule. Les informations structurelles souvent perdus lors de la transformation en CNF peuvent s’avérer utiles pour la résolution [5,6].

Dans ce papier, nous proposons un nouveau prétraitement pour la résolution d’instances SAT, qui extrait et exploite certaines structures cachées dans la CNF. Cette technique se base sur une utilisation originale de la procédure de propagation de contraintes (BCP). Alors que BCP est souvent utilisée pour produire des littéraux impliqués ou équivalents, nous montrons qu’il est possible d’étendre son utilité en produisant des

nouvelles fonctions booléennes de la forme $y = f(x_1, \dots, x_n)$ où f est un opérateur parmi $\{\vee, \wedge\}$ avec y et x_i représentant les variables booléennes de la formule initiale. Ces fonctions nous permettent de détecter un sous-ensemble de variables indépendantes qui peuvent ensuite être exploités par des solveurs SAT.

Ce papier étend de manière significative des résultats préliminaires de [7] dans le sens où plus de fonctions et plus de variables dépendantes sont détectées et ce pour de nombreuses classes d'instances. Malheureusement dans cet ensemble de dépendances fonctionnelles, des cycles peuvent apparaître, et la détection d'un ensemble coupe-cycle minimal est un problème NP-Difficile. Pour ce faire, nous proposons des heuristiques efficaces pour construire un ensemble coupe-cycle de taille raisonable. Cela nous permet de déterminer un ensemble de variables « essentielles » dont le sens où toute affectation possible de ces variables détermine le reste de la formule. Cette ensemble de variables composé de variables indépendantes et de variable de l'ensemble coupe-cycle peut être considéré comme une forme de « *strong backdoor* » tel que définit par Williams-etal [8].

Le papier est organisé de la manière suivante. Après quelques définitions préliminaires, nous présentons les fonctions booléennes (portes) ainsi que leurs propriétés. Nous montrons ensuite comment il est possible de détecter plus de fonctions booléennes que dans [7] en utilisant la propagation de contraintes. Puis, nous présentons une méthode permettant de construire à partir de la formule considérée et des fonctions booléennes détectées un ensemble de variables essentielles, dans le but de réduire l'espace de recherche. Nous présentons ensuite les résultats expérimentaux obtenus sur de nombreuses instances réelles issus des dernières compétitions internationales, montrant l'intérêt de notre approche. Finalement quelques pistes de recherche sont avancées en conclusion.

2 Définitions préliminaires

Soit $\mathcal{L}_{\mathcal{P}}$ un langage de formules propositionnelles, construit de manière usuel en utilisant les connecteurs classique ($\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow$) et \mathcal{P} un ensemble de variables propositionnelles.

Une *formule CNF* Σ est un ensemble (interprété comme une conjonction) de *clauses*, où une clause est un ensemble (interprété comme une disjonction) de *littéraux*. Un littéral est une variable propositionnelle positive ou négative. On note $\mathcal{V}(\Sigma)$ (resp. $\mathcal{L}(\Sigma)$) l'ensemble des variables (resp. littéraux) apparaissant dans Σ . Une *clause unitaire* est une clause formée d'un unique littéral, appelé *littéral unitaire*.

En plus de ces notations, nous définissons la négation d'un ensemble de littéraux $\neg\{l_1, \dots, l_n\}$ comme l'ensemble correspondant aux littéraux opposés $\{\neg l_1, \dots, \neg l_n\}$.

Une *interprétation* d'une formule booléenne est une affectation des valeurs $\{vrai, faux\}$ à ses variables. Un *modèle* d'une formule CNF est une interprétation qui satisfait la formule, c'est-à-dire qui satisfait chaque clause de la CNF. Une clause est satisfaite si au moins un de ses littéraux est à *vrai*. Le problème SAT revient à prouver l'existence ou non d'un modèle pour une formule CNF.

Soit c_1 une clause contenant un littéral a et c_2 une clause contenant le littéral opposé $\neg a$, la *résolvante* de c_1 avec c_2 est la disjonction de tous les littéraux de c_1 et c_2 privé de a et $\neg a$. Une résolvante est dite *tautologique* si elle contient des littéraux opposés.

Rappelons qu'une formule booléenne peut être transformée en temps linéaire en une formule CNF équivalente pour la satisfiabilité (par ajout de variables supplémentaires). La plupart des algorithmes de résolution travaillent sur la forme clausale où la structure du problème se retrouve cachée. Par la suite une formule CNF sera représentée par un ensemble de fonctions (portes) booléennes.

3 Portes booléennes

Une *porte (booléenne)* est une expression de la forme $y = f(x_1, \dots, x_n)$, où f est un opérateur logique parmi $\{\vee, \wedge, \Leftrightarrow\}$ et où y et x_i sont des littéraux propositionnels. Ces portes se traduisent sous forme CNF de la manière suivante :

- $y = \wedge(x_1, \dots, x_n)$ peut être représenté par l'ensemble de clauses $\{(y \vee \neg x_1 \vee \dots \vee \neg x_n), (\neg y \vee x_1), \dots, (\neg y \vee x_n)\}$, traduisant le fait que la valeur de y est entièrement déterminée par les valeurs de x_i avec $i \in [1..n]$;
- $y = \vee(x_1, \dots, x_n)$ peut être représenté par l'ensemble de clauses $\{\neg y \vee x_1 \vee \dots \vee x_n, y \vee \neg x_1, \dots, y \vee \neg x_n\}$;
- $y = \Leftrightarrow(x_1, \dots, x_n)$ représente la *chaîne d'équivalences* (aussi appelé *formule biconditionnelle*) $y \Leftrightarrow x_1 \Leftrightarrow \dots \Leftrightarrow x_n$, qui est équivalente à 2^n clauses.

Une clause est représentable par une porte de la forme *vrai* $= \vee(x_1, \dots, x_n)$. De plus, une porte $\neg y = \wedge(x_1, \dots, x_n)$ (resp. $\neg y = \vee(x_1, \dots, x_n)$) est équivalente à $y = \vee(\neg x_1, \dots, \neg x_n)$ (resp. $y = \wedge(\neg x_1, \dots, \neg x_n)$). Par ailleurs, la propriété sur les chaînes d'équivalences énonçant qu'une chaîne d'équivalences possédant un nombre pair de littéraux négatifs (resp. impair) est équivalente à la chaîne formée des mêmes littéraux mais tous positifs (resp. excepté un), permet de réécrire chaque porte de la forme $y = \Leftrightarrow(x_1, \dots, x_n)$ en une porte où y est un littéral positif. Par exemple, $\neg y = \Leftrightarrow(\neg x_1, x_2, x_3)$ est équivalente à $y = \Leftrightarrow(x_1, x_2, x_3)$ et $\neg y = \Leftrightarrow(\neg x_1, x_2, \neg x_3)$ est équivalente par exemple à $y = \Leftrightarrow(x_1, x_2, \neg x_3)$.

Une variable propositionnelle y (resp. x_1, \dots, x_n) est une *variable de sortie* (resp. sont *des variables d'entrée*) d'une porte de la forme $y = f(x'_1, \dots, x'_n)$ avec $x'_i \in \{x_i, \neg x_i\}$.

Une variable propositionnelle z est une *variable de sortie (ou dépendante)* d'un ensemble de portes ssi z est une variable de sortie d'au moins une porte de cet ensemble. Une *variable d'entrée (ou indépendante)* d'un ensemble de portes est une variable qui n'est variable de sortie d'aucune porte de cet ensemble.

Une porte est satisfaite sous une interprétation ssi la partie gauche et droite de la porte sont toutes les deux à *vrai* ou à *faux* sous cette interprétation. Une interprétation satisfait un ensemble de portes ssi chaque porte de cet ensemble est satisfaite sous cette interprétation. Une telle interprétation est alors appelée modèle de cet ensemble de portes.

4 De la CNF aux portes

En pratique, nous souhaitons trouver une représentation d'une formule CNF Σ utilisant les portes et nous permettant de *maximiser* le nombre de variables dépendantes dans le but de réduire la complexité pour décider de la satisfaisabilité de Σ . En réalité, nous décrivons une technique qui extrait des portes pouvant être déduites à partir de Σ , et qui *couvre* un sous-ensemble de clauses de Σ . Dans le but d'obtenir une représentation uniforme, les clauses restantes de Σ sont représentées par des portes « ou » de la forme $vrai = \vee(x_1, \dots, x_n)$.

Plus formellement, supposons qu'un ensemble de clauses $Cl(G)$ correspondant à un ensemble de portes G soit une conséquence logique d'une CNF Σ , alors l'ensemble $\Sigma_{non-couvert(G)}$ des clauses non couvertes de Σ par rapport à G est l'ensemble de clauses : $\Sigma \setminus Cl(G)$.

Par conséquent, $\Sigma \equiv \Sigma_{non-couvert(G)} \cup Cl(G)$.

Trivialement, on peut voir que les clauses additionnelles $Cl(G) \setminus \Sigma$ peuvent jouer un rôle important pour la déduction ou la recherche de satisfaisabilité.

De plus, il est clair que connaître les variables de sortie peut jouer un rôle important pour la décider de la satisfaisabilité d'une formule CNF. En effet, la valeur de vérité d'une variable y d'une porte dépend de la valeur de vérité de ses variables d'entrées x_i . La valeur de vérité de cette variable de sortie peut être obtenu par propagation et peut ne pas être pris en compte dans les heuristiques de branchement des algorithmes de type DPLL [9]. Dans le cas général, déterminer n' variables de sortie à partir de la représentation sous forme de portes d'une CNF comptant n variables permet de réduire le nombre d'interprétations à explorer de 2^n à $2^{n-n'}$. Évidemment, la réduction de l'arbre de recherche augmente avec le nombre de variables dépendantes détectées.

Malheureusement, pour obtenir une telle réduction de l'arbre de recherche, les problèmes suivants sont soulevés :

- Déduire des portes d'une formule CNF, dans le cas général, est trop coûteux, à moins de fournir quelques critères simples pour l'extraction. En effet, montrer que $y = f(x_1, \dots, x_i)$ (où y, x_1, \dots, x_i appartiennent à Σ) est une porte de Σ , est un problème coNP-complet.
- Lorsque l'ensemble des portes détectées contient des définitions croisées (comme $y = f(x, t)$ et $x = g(y, z)$), brancher sur les variables d'entrées (indépendantes) n'est pas suffisant pour déterminer la valeur de vérité des variables dépendantes. Traiter ces définitions croisées coïncide avec le problème NP-Difficile de détection d'ensemble coupe-cycle minimal dans un graphe.

Dans ce papier, nous traitons ces deux aspects : le premier en restreignant la déduction à la déduction par propagation de contraintes seulement ; le second en utilisant des heuristiques efficaces de détection d'un ensemble coupe-cycle de taille raisonnable sur un graphe biparti représentant l'ensemble de portes.

Rappelons tout d'abord quelques définitions sur la propagation de contraintes.

5 Propagation de contraintes booléennes (BCP)

La propagation des contraintes booléennes ou résolution unitaire est la plus utilisée et la plus utile des techniques pour SAT.

Soit Σ une formule CNF, $BCP(\Sigma)$ est la formule CNF obtenue en *propageant* tous les littéraux unitaires de Σ . Propager un littéral unitaire l de Σ consiste à supprimer d'une part toutes les clauses c de Σ tel que $l \in c$ et d'autre part de remplacer toutes les clauses c' de Σ tel que $\neg l \in c'$ par $c' \setminus \{\neg l\}$. La formule CNF obtenue de cette manière est équivalente à Σ du point de vue de la satisfiabilité.

L'ensemble des littéraux unitaires propagés de Σ utilisant BCP est noté $UP(\Sigma)$. Évidemment, nous avons $\Sigma \models UP(\Sigma)$. BCP est une forme de résolution et est applicable en temps linéaire. Cette méthode est complète pour les formules de Horn. De plus, BCP est utilisée dans de nombreux solveurs SAT comme prétraitement afin d'en déduire des informations intéressantes comme des littéraux impliqués [10] et des littéraux équivalents [11,12]. Des traitements locaux basés sur cette technique permettent également de fournir des choix intéressants de variables (comme l'heuristique UP [13]).

Dans la suite, nous montrons que cette technique peut être étendue afin de déduire des dépendances fonctionnelles.

6 BCP et dépendances fonctionnelles

En réalité, BCP peut être utilisée afin de détecter des dépendances fonctionnelles. Le résultat principal de ce papier consiste à exploiter de manière pratique la propriété suivante : les portes peuvent être détectées en utilisant simplement BCP, alors que dans le cas général déterminer si une porte est une conséquence logique d'une formule CNF est un problème coNP-complet.

Propriété 1. Soit Σ une formule CNF, $l \in \mathcal{L}(\Sigma)$, et $c \in \Sigma$ tel que $l \in c$. Si $c \setminus \{l\} \subset \neg UP(\Sigma \wedge l)$ alors $\Sigma \models l = \wedge(\neg\{c \setminus \{l\}\})$.

Démonstration. Soit $c = \{l, \neg l_1, \neg l_2, \dots, \neg l_m\} \in \Sigma$ tel que $c \setminus \{l\} = \{\neg l_1, \neg l_2, \dots, \neg l_m\} \subset \neg UP(\Sigma \wedge l)$. La fonction booléenne $l = \wedge(\neg\{c \setminus \{l\}\})$ peut être réécrite comme $l = \wedge(l_1, l_2, \dots, l_m)$. Afin de prouver que $\Sigma \models l = \wedge(l_1, l_2, \dots, l_m)$, nous devons montrer que chaque modèle de Σ , est aussi un modèle de $l = \wedge(l_1, l_2, \dots, l_m)$. Soit I un modèle de Σ , alors

1. soit l est à *vrai* dans I : I est aussi un modèle de $\Sigma \wedge l$. Comme $\{\neg l_1, \neg l_2, \dots, \neg l_m\} \subset \neg UP(\Sigma \wedge l)$, nous avons $\{l_1, l_2, \dots, l_m\} \subset UP(\Sigma \wedge l)$, alors $\{l_1, l_2, \dots, l_m\}$ sont à *vrai* dans I . Donc, I est aussi un modèle de $l = \wedge(l_1, l_2, \dots, l_m)$;
2. soit l est à *faux* dans I : comme $c = \{l, \neg l_1, \neg l_2, \dots, \neg l_m\} \in \Sigma$ alors I satisfait $c = \{\neg l_1, \neg l_2, \dots, \neg l_m\} \in \Sigma$. Donc, au moins un littéral $l_i, i \in \{1, \dots, m\}$ est à *vrai* dans I . Donc, I est aussi un modèle de $l = \wedge(l_1, l_2, \dots, l_m)$

Clairement, en fonction du signe du littéral l , les portes « et » et « ou » peuvent être détectés. Par exemple, la porte « et » $\neg l = \wedge(l_1, l_2, \dots, l_n)$ est équivalente à la porte « ou » $l = \vee(\neg l_1, \neg l_2, \dots, \neg l_n)$. Cette propriété couvre aussi les équivalences binaires car $a = \wedge(b)$ est équivalent à $a \Leftrightarrow b$.

En réalité, cette propriété nous permet de détecter de nouvelles portes par rapport à la méthode syntaxique utilisée dans [7]. Illustrons la par un exemple.

Exemple 1. Soit $\Sigma_1 \supseteq \{(y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3), (\neg y \vee x_1), (\neg y \vee x_2), (\neg y \vee x_3)\}$.

Dans [7], Σ_1 peut être représenté par un graphe où chaque sommet est une clause et chaque arête indique s'il existe une résolvente tautologique entre deux clauses du graphe. Chaque composante connexe du graphe peut représenter une porte. Comme on peut le voir ces quatre clauses appartiennent à une même composante connexe. C'est une condition nécessaire pour qu'un tel sous-ensemble de clauses encode une porte. Une fois cet ensemble de clauses identifié (constitué de clauses apparaissant dans une composante connexe), on doit ensuite déterminer de manière syntaxique si l'on est en présence d'une porte et identifier cette porte (« et » ou « ou »). Cette détermination se fait en temps polynômial. Dans l'exemple précédent, nous avons $y = \wedge(x_1, x_2, x_3)$.

Considérons maintenant l'exemple suivant,

Exemple 2. $\Sigma_2 \supseteq \{y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3, \neg y \vee x_1, \neg x_1 \vee x_4, \neg x_4 \vee x_2, \neg x_2 \vee x_5, \neg x_4 \vee \neg x_5 \vee x_3\}$.

Clairement la représentation sous forme de graphe de cet ensemble de clauses est différente et la technique utilisée précédemment ne nous permet pas de découvrir la porte $y = \wedge(x_1, x_2, x_3)$. La condition nécessaire mais non suffisante n'est plus satisfaite.

Maintenant, selon la propriété 1, les deux portes « et » des exemples 1 et 2 sont détectées. En effet, dans l'exemple 1, $UP(\Sigma_1 \wedge y) = \{x_1, x_2, x_3\}$ et $\exists c \in \Sigma_1, c = (y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3)$ tel que $c \setminus \{y\} \subset \neg UP(\Sigma_1 \wedge y)$. De plus, dans l'exemple 2, $UP(\Sigma_2 \wedge y) = \{x_1, x_4, x_2, x_5, x_3\}$ et $\exists c' \in \Sigma_2, c' = (y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3)$ tel que $c' \setminus \{y\} \subset \neg UP(\Sigma_2 \wedge y)$.

Par conséquent, une technique de recherche de portes consiste à vérifier la propriété 1 pour chaque littéral apparaissant dans Σ .

Ayant déterminé l'ensemble de portes, une autre étape consiste à trouver les variables dépendantes de la formule originale. Une porte nous permet de déterminer une variable dépendante par rapport aux variables d'entrées qui sont indépendantes. Cependant si plusieurs portes partagent un même littéral, la caractérisation des variables dépendantes n'est plus la même. En effet, on peut voir apparaître des cycles comme le montre l'exemple suivant.

Exemple 3. $\Sigma_3 \supseteq \{x = \wedge(y, z), y = \vee(x, \neg t)\}$.

Clairement, Σ_3 contient un cycle. En effet, x dépend des variables y et z , alors que y dépend des variables x et t . Si une seule porte est considérée, alors affecter des valeurs aux variables d'entrées permet de déterminer la valeur de la variable de sortie. Mais dans l'exemple 3, affecter une valeur aux variables qui ne sont pas considérées comme dépendantes n'est pas suffisant (dans tous les cas) pour déterminer la valeur des variables de sorties. Dans l'exemple, affecter une valeur à z et t n'est pas suffisant pour déterminer la valeur de x et de y . Néanmoins, dans l'exemple, si l'on affecte une valeur à une variable du cycle, x par exemple, alors la valeur de y est déterminé dans tous les cas. Une telle variable est appelée *variable coupe-cycle*. Afin de déterminer un sous-ensemble des variables qui une fois affectées détermineront la valeur de toutes les autres, il est indispensable de supprimer ces formes de cycles. Un tel ensemble est appelé *strong backdoor* dans [8], que nous appellerons simplement *backdoor* pour simplifier. Dans l'exemple 3, un backdoor correspond à l'ensemble $\{x\} \cup \{z, t\}$. Dans

ce contexte, le backdoor est constitué des variables indépendantes et des variables du coupe-cycle. Trouver l'ensemble minimal de variables coupe-cycle dans un graphe est un problème NP-Difficile. Nous traitons de ce sujet dans le paragraphe suivant.

7 Recherche de variables dépendantes

Par la suite, nous considérons une représentation des portes détectées sous forme de graphe. Plus formellement, un ensemble de portes peut être représenté par un graphe biparti $G = (O \cup I, E)$ défini comme suit :

- À chaque porte est associé deux sommets. Le premier $o \in O$ représente la variable de sortie de la porte et le second $i \in I$ représente l'ensemble des variables d'entrées. Le nombre total de sommets est donc inférieur à $2 \times \#gates$, où $\#gates$ est le nombre de portes.
- Pour chaque porte, une arête (o, i) entre les deux sommets o et i représentant la partie gauche et droite de la porte est créée. Des arêtes supplémentaires sont créées entre $o \in O$ et $i \in I$ si une variable de sortie associée au sommet o appartient à l'ensemble des variables d'entrées du sommet i .

Trouver le *plus petit* sous-ensemble V' de O tel que le sous-graphe $G' = (V' \cup I, E')$ est acyclique est un problème NP-Difficile.

En pratique, tout sous-ensemble V' rendant le graphe acyclique associé à l'ensemble des variables indépendantes, permet de déterminer toutes les autres. Lorsque V' est de taille c et l'ensemble des variables dépendantes est de taille d , alors l'espace de recherche est réduit de 2^n à $2^{n-(d-c)}$, où n est le nombre total de variables de la formule CNF.

Nous devons trouver un compromis entre la taille de V' , et le gain en temps pour la résolution ensuite.

Dans la suite, nous proposons deux heuristiques pour trouver un ensemble coupe cycle V' . La première s'appelle *Maxdegré*. Cette heuristique consiste à construire V' de manière incrémentale en sélectionnant le sommet de plus haut degré jusqu'à ce que le graphe soit acyclique.

La seconde est appelée *MaxdegréCycle*. Elle consiste à construire de manière incrémentale V' en sélectionnant le sommet de plus haut degré appartenant à un cycle. Cette heuristique garantit qu'à chaque fois qu'une variable est choisie, au moins un cycle est supprimée.

Dans le paragraphe suivant, nous présentons et discutons des résultats expérimentaux de notre approche. Ces résultats regroupent à la fois les portes détectées et le calcul des variables du coupe-cycle dans le but de nous fournir un partitionnement des variables (indépendantes et dépendantes). Deux stratégies sont explorées. Dans la première, à chaque fois qu'une porte est trouvée les clauses la constituant sont supprimées de la CNF initiale. Dans la seconde, les clauses participant au codage d'une porte ne sont supprimées qu'à la fin de la détection des portes. En terme de nombre et de type de portes détectées, il est clair que la première approche dépend fortement de l'ordre dans lequel la propagation des littéraux va être effectuée. En ce qui concerne la seconde approche, l'ordre de propagation ne peut pas influencer sur les portes détectées. Ces deux

Classes d'Instances		Technique proposée dans [7] #G	Notre approche		
Nom	(#Inst.,#V[<i>min</i> -Max],#C[<i>min</i> -Max])		Sans Suppression #G	Avec Suppression #G #C supp.	
Blocks	(3,484[283-758],27423[9690-47820])	10[3]	236[134]	18[5]	271[142]
Logistics	(8,994[116-3016],12706[953-50457])	380[265]	437[417]	169[213]	630[585]
Pipe	(6,1642[834-2577],18624[6695-33270])	1312[679]	1407[697]	1240[639]	13898[9083]
Facts	(13,3178[2218-4315],48737[22539-90646])	713[147]	1601[541]	497[170]	1731[510]
Parity	(30,1044[64-3176],3614[254-10325])	568[828]	510[594]	328[455]	663[870]
Qg	(10,969[512-1331],33747[9685-64054])	310[91]	1828[652]	298[80]	1708[601]
Ca	(7,637[26-2282],1835[70-6586])	419[547]	459[592]	414[542]	1233[1615]
Dp	(11,1427[213-3193],3580[376-8308])	1117[856]	1468[1211]	915[812]	2534[2298]
Bmc2	(5,1952[316-4089],6908[1002-13531])	895[714]	1025[850]	744[623]	2082[1824]
Rand	(6,2217[2000-2500],6568[5921-7401])	2133[236]	2444[381]	2103[252]	6212[692]
Ezfact	(40,1441[193-3073],9169[1113-19785])	40[18]	268[127]	68[33]	68[33]
Med	(3,761[341-1159],20154[5556-36291])	66[32]	316[162]	14[5]	319[164]
Avg-checker	(4,917[648-1188],28661[17087-40441])	324[105]	1098[375]	304[101]	1092[373]
nw/nc/fw	(13,3997[2756-5074],15829[10886-20123])	89[40]	468[136]	125[38]	125[38]
Am	(4,2011[433-4264],6925[1458-14751])	989[835]	772[585]	393[276]	927[625]
Cnf	(2,2424[2424-2424],14812[14812-14812])	2336[0]	3280[0]	2301[6]	13703[149]

TAB. 1. #G : Nombre de portes détectées (moyenne[écart-type])

stratégies sont comparées en terme de portes détectées, de taille de l'ensemble coupe-cycle, du nombre de variables dépendantes et du nombre de clauses non couvertes.

8 Résultats expérimentaux

Le prétraitement proposé dans cet article a été implémenté en C sous Linux et peut être récupéré à l'adresse suivante :

<http://www.cril.univ-artois.fr/~ostrowski/Binaries/llsatpreproc>.

Toutes les expérimentations ont été conduites sur un Pentium IV, 2.4 Ghz. La description des instances testées peuvent être trouvées sur le site web SATLib (<http://www.satlib.org>).

Nous avons comparé notre approche à la technique décrite dans [7] sur des instances issues des dernières compétitions SAT [14,15]. Ces instances sont de provenances diverses : model-checking, VLSI, planification ... Les résultats complets des expérimentations conduites sont disponibles à l'adresse :

<http://www.cril.univ-artois.fr/~ostrowski/result-llsatpreproc.ps>.

Afin d'obtenir des tables plus synthétiques, nous avons regroupé les instances d'une même famille sur une seule ligne. Pour chaque classe d'instances, nous indiquons le nombre d'instances testées, les nombres moyens, minimaux et maximaux de variables et de clauses. Pour chaque résultat, nous indiquons la moyenne et l'écart-type (entre crochets) obtenue pour une famille d'instances.

La table 1, regroupe les résultats obtenus en terme du nombre de portes détectées (#G) pour l'approche décrite dans [7] et pour les approches proposées. Nous avons distingué deux approches : l'une où aucune suppression de clauses n'est opérée durant la détection des portes et l'autre où les clauses couvertes par chacune des portes détectées sont supprimées au fur et à mesure de la détection. Les résultats montrent

Classe d'Instance	(#V[<i>min</i> - <i>Max</i>])	<i>Maxdegré</i>			<i>MaxdegréCycle</i>		
		# <i>D</i>	# <i>CS</i>	# <i>B</i>	# <i>D</i>	# <i>CS</i>	# <i>B</i>
Blocks	(484[283-758])	38[13]	198[123]	353[215]	39[9]	197[124]	352[216]
Logistics	(994[116-3016])	113[158]	245[218]	441[532]	143[164]	214[194]	410[522]
Pipe	(1642[834-2577])	980[768]	265[219]	582[201]	764[449]	481[192]	798[348]
Facts	(3178[2218-4315])	738[237]	813[256]	1964[604]	487[124]	1064[362]	2216[623]
Parity	(1044[64-3176])	243[388]	84[46]	573[528]	287[410]	40[21]	528[505]
Qg	(969[512-1331])	303[202]	228[236]	228[236]	11[6]	521[194]	521[194]
Ca	(637[26-2282])	290[434]	130[142]	344[403]	265[341]	155[206]	369[481]
Dp	(1427[213-3193])	513[463]	451[485]	725[625]	551[496]	412[343]	686[498]
Bmc2	(1952[316-4089])	662[716]	27[22]	886[874]	660[696]	30[10]	888[893]
Rand	(2217[2000-2500])	1777[301]	357[339]	440[343]	1152[134]	981[111]	1064[115]
Ezfact	(1441[193-3073])	28[35]	66[45]	1370[1073]	55[27]	39[18]	1343[1060]
Med	(761[341-1159])	205[102]	110[72]	110[72]	14[4]	302[157]	302[157]
Avg-checker	(917[648-1188])	209[357]	606[283]	606[283]	276[94]	539[187]	539[187]
nw/nc/fw	(3997[2756-5074])	39[48]	151[47]	3899[854]	94[24]	96[23]	3844[855]
Am	(2011[433-4264])	327[263]	97[68]	413[241]	298[206]	126[99]	441[287]
Cnf	(2424[2424-2424])	472[564]	1801[564]	1953[564]	1170[2]	1103[2]	1255[2]

TAB. 2. Taille du backdoor sans suppression de clauses

clairement que notre nouvelle approche permet d'identifier plus de portes. Par ailleurs, il n'est pas surprenant de détecter moins de portes lorsque l'on retire les clauses au fur et à mesure.

Dans la table 2, nous avons pris l'option de ne pas supprimer les clauses. Cette table a pour objectif de comparer les deux heuristiques coupe-cycle : *Maxdegré* et *MaxdegréCycle*. Pour chaque classe d'instances sont fournis : la moyenne du nombre de variables dépendantes (#*D*), la taille moyenne de l'ensemble coupe-cycle (#*CS*) et la taille moyenne du backdoor (#*B*). À chaque moyenne, l'écart-type est associé entre crochets. Il est à noter que sur certaines classes d'instances, le backdoor ne représente que seulement 10% du nombre de variables !

Dans la table 3, nous avons pris l'option de supprimer les clauses. Le nombre de portes détectées est bien inférieur à l'approche sans suppression. En contre-partie, la taille de l'ensemble coupe-cycle est généralement plus petit.

En conséquence, aucune option n'est préférable dans le cas général. En effet, trouver un petit backdoor, dépend à la fois de la classe d'instance et des options considérées.

Cependant, dans la plupart des cas, l'option de supprimer les clauses et l'heuristique *MaxdegréCycle* engendre un backdoor plus réduit.

Nous essayons actuellement d'intégrer de manière efficace ce prétraitement dans un solveur SAT performant (e.g. *Zchaff* [16]). L'objectif étant de guider le solveur dans ces choix de branchements en fonction du backdoor identifié. Pour l'instant ce prétraitement n'as pas du tout été optimisé. Néanmoins, il est très efficace en pratique (**moins de 1 seconde dans la plupart des cas**). Cette dernière remarque explique la non présence d'une colonne temps dans les différentes tables.

9 Perspectives

Une première piste possible concerne l'utilisation des clauses découvertes. Notre représentation de formules sous la forme de portes peut se révéler efficace pour la réso-

classe d'Instances	(#V[<i>min</i> - <i>Max</i>])	<i>Maxdegré</i>			<i>MaxdegréCycle</i>		
		# <i>D</i>	# <i>CS</i>	# <i>B</i>	# <i>D</i>	# <i>CS</i>	# <i>B</i>
Blocks	(484[283-758])	18[4]	0[0]	373[219]	18[4]	0[0]	373[219]
Logistics	(994[116-3016])	135[147]	25[48]	419[539]	152[178]	7[13]	401[509]
Pipe	(1642[834-2577])	1020[735]	219[215]	543[223]	956[513]	282[124]	606[283]
Facts	(3178[2218-4315])	488[127]	0[0]	2214[621]	488[127]	0[0]	2214[621]
Parity	(1044[64-3176])	318[426]	0[0]	497[480]	318[426]	0[0]	497[480]
Qg	(969[512-1331])	122[99]	138[87]	410[189]	181[60]	80[25]	351[140]
Ca	(637[26-2282])	317[433]	94[113]	317[392]	302[388]	109[151]	332[434]
Dp	(1427[213-3193])	724[643]	149[151]	513[357]	728[641]	145[143]	509[353]
Bmc2	(1952[316-4089])	680[706]	1[1]	868[883]	680[705]	1[1]	868[884]
Rand	(2217[2000-2500])	1591[418]	495[396]	625[401]	1200[129]	886[102]	1016[111]
Ezfact	(1441[193-3073])	48[23]	10[5]	1350[1064]	49[23]	9[5]	1349[1064]
Med	(761[341-1159])	14[4]	0[0]	302[157]	14[4]	0[0]	302[157]
Avg-checker	(917[648-1188])	302[100]	0[0]	512[181]	302[100]	0[0]	512[181]
nw/nc/fw	(3997[2756-5074])	73[14]	40[22]	3864[857]	95[24]	18[10]	3842[856]
Am	(2011[433-4264])	367[254]	0[0]	373[239]	367[254]	0[0]	373[239]
Cnf	(2424[2424-2424])	1988[12]	285[12]	437[12]	2210[6]	63[6]	215[6]

TAB. 3. Taille du backdoor avec option de suppression

lution du problème de la satisfiabilité. Pour illustrer cette idée, considérons de nouveau l'exemple 2. De la formule Σ , on en déduit la porte $y = \wedge(x_1, x_2, x_3)$. La forme clause de cette porte est donné par $\{y \vee \neg x_1 \vee \neg x_2 \vee \neg x_3, \neg y \vee x_1, \neg y \vee x_2, \neg y \vee x_3\}$.

Clairement, les clauses additionnelles $\{\neg y \vee x_2, \neg y \vee x_3\}$ sont des résolvantes de Σ , qui ne peuvent être obtenu qu'au bout de deux et six étapes de résolution respectivement. Donc, la représentation des portes de Σ induit des résolvantes binaires non triviales qui peuvent rendre la déduction ou la résolution plus facile. Prendre en compte cette caractéristique dans une base de clauses ou de portes serait une piste intéressante pour réduire l'espace de recherche. Certaines portes découvertes représentent aussi des équivalences binaires ($x \leftrightarrow y$) et les remplacer permettrait d'obtenir une réduction importante de l'espace de recherche.

Une autre piste possible concerne l'analyse du graphe obtenu. Comment par exemple utiliser des techniques de décomposition sur celui-ci.

Enfin, afin de réduire encore la taille du backdoor, nous étudions comment exploiter les parties traitables de la formule (par exemple les formules de horn ou horn-renommable).

10 Conclusions

Clairement, nos expérimentations sont assez encourageantes. Détecter un ensemble de variables dépendantes peut se faire à moindre coût. Des cycles dans le graphe apparaissent mais peuvent être supprimés efficacement. Nous implémentons actuellement ce prétraitement dans un solveur SAT efficace. De plus, nous pensons que l'étude des cycles et des variables dépendantes est essentiel pour la compréhension des problèmes SAT difficiles.

11 Remerciements

Ce travail à été soutenu en partie par le CNRS, le FEDER, l'IUT de Lens et le Conseil Régional du Nord/Pas-de-Calais.

Références

1. Oler, S. : Tuning sat checkers for bounded model checking. In : Proceedings of Computer Aided Verification (CAV'00). (2000)
2. Giunchiglia, E., Maratea, M., Tacchella, A., Zambonin, D. : Evaluating search heuristics and optimization techniques in propositional satisfiability. In : Proceedings of International Joint Conference on Automated Reasoning (IJCAR'01), Siena (2001)
3. Zhang, L., Madigan, C., Moskewicz, M., Malik, S. : Efficient conflict driven learning in a boolean satisfiability solver. In : Proceedings of ICCAD'2001, San Jose, CA (USA) (2001) 279–285
4. Dubois, O., Dequen, G. : A backbone-search heuristic for efficient solving of hard 3–sat formulae. In : Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01). Volume 1., Seattle, Washington (USA) (2001) 248–253
5. Rauzy, A., Saïs, L., Brisoux, L. : Calcul propositionnel : vers une extension du formalisme. In : Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99), Lyon (1999) 189–198
6. Kautz, H.A., McAllester, D., Selman, B. : Exploiting variable dependency in local search. In : Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97", Nagoya (Japan) (1997)
7. Ostrowski, R., Grégoire, E., Mazure, B., Saïs, L. : Recovering and exploiting structural knowledge from cnf formulas. In : Eighth International Conference on Principles and Practice of Constraint Programming (CP'2002), Ithaca (N.Y.), LNCS 2470, Springer Verlag (2002) 185–199
8. Williams, R., Gomez, C.P., Selman, B. : Backdoors to typical case complexity. In : Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03). (2003) 1173–1178
9. Davis, M., Logemann, G., Loveland, D. : A machine program for theorem proving. Journal of the Association for Computing Machinery **5** (1962) 394–397
10. Dubois, O., André, P., Boufkhad, Y., Carlier, J. : Sat versus unsat. In Johnson, D., Trick, M., eds. : Second DIMACS Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society (1996) 415–436
11. Brisoux, L., Saïs, L., Grégoire, E. : Recherche locale : vers une exploitation des propriétés structurelles. In : Actes des Sixièmes Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets(JNPC'00), Marseille (2000) 243–244
12. Le Berre, D. : Exploiting the real power of unit propagation lookahead. In : Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001), Boston University, Massachusetts, USA (2001)
13. Li, C.M., Anbulagan : Heuristics based on unit propagation for satisfiability problems. In : Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya (Japan) (1997) 366–371

14. SAT 2002 : Fifth international symposium on theory and applications of satisfiability testing (2002) <http://gauss.eecs.uc.edu/Conferences/SAT2002/>.
15. SAT 2003 : Sixth international symposium on theory and applications of satisfiability testing (2003) <http://www.mrg.dist.unige.it/events/sat03/>.
16. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S. : Chaff : Engineering an efficient sat solver. In : Proceedings of 38th Design Automation Conference (DAC01). (2001)
17. Bacchus, F., Winter, J. : Effective preprocessing with hyper-resolution and equality reduction. In : Sixth International Symposium on Theory and Applications of Satisfiability Testing (SAT'03). (2003)
18. Jarvisalo, M., Junttila, T., Niemelä, I. : Unrestricted vs restricted cut in a tableau method for Boolean circuits. In : AI&M 2004, 8th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, USA (2004)