Une approche symbolique pour les formules booléennes quantifiées

Gilles Audemard **Lakhdar Saïs** *

CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18
F-62307 Lens Cedex France

{audemard,sais}@cril.univ-artois.fr

Résumé

Depuis quelques années, la résolution des formules booléennes quantifiées (QBF) est devenu un domaine de recherche important et attractif. En effet, de nombreuses classes de problèmes peuvent être formulées efficacement en des instances QBF (planification, raisonnement non monotone, model checking). Beaucoup de solveurs ont été proposé. La majorité d'entre eux explorent un arbre de recherche en utilisant des techniques basés sur la procédure DLL. Or, les quantificateurs des formules QBF imposent un ordre partiel d'affectation des variables. Pour passer outre cette imposition, nous proposons une nouvelle approche symbolique QBDD(SAT)). Elle utilise de façon originale les diagrammes de décision binaires pour représenter l'ensemble des modèles (ou impliquants premiers) de la formule booléenne en utilisant des algorithmes de recherches pour SAT. Deux extensions améliorent notre approche. Tout d'abord, nous introduisons un opérateur de réduction sur les BDD afin de limiter leur taille et de pouvoir répondre à la question de la validité des formules QBF. Ensuite, grâce à des *nogoods* générés à partir du BDD, des coupures sont réalisés dans l'arbre de recherche. En utilisant des techniques basées sur DLL (resp. recherche stochastique), notre approche peut facilement donner lieu à un solveur complet QBDD(DLL) (resp. incomplet QBDD(LS)). Les premières expérimentations montrent de bonnes performances de ces deux solveurs sur certaines classes d'instances de l'évaluation QBF03.

*Ce travail est en partie supporté par l'IUT de Lens, le CNRS et la Région Nord/Pas-de-Calais sous le programme TAC.

1 Introduction

Depuis quelques années, la résolution des formules booléennes quantifiées (QBF) est un domaine de recherche en pleine ébullition. Cet intérêt peut être expliqué par plusieurs facteurs. Tout d'abord, de nombreux problèmes en intelligence artificielle (planification, raisonnement non monotone, vérification formelle, ...) peuvent se réduire à des formules booléennes quantifiées. De plus, le problème de la validité des QBF est considéré comme le problème central de la classe de complexité PSPACE. Enfin, les progrès incessants réalisés sur la résolution du problème SAT permettent maintenant de s'attaquer à des formules de plus en plus grandes et difficiles.

Récemment, de nombreux solveurs QBF ont été proposé (voir par exemple [12, 18, 13, 11]). La plupart d'entre eux, étendent les derniers résultats obtenus sur la résolution du problème SAT. Ceci n'a rien de surprenant puisque les formules QBF sont une extension naturelle du problème SAT où les variables booléennes sont quantifiées universellement ou existentiellement. Ces solveurs considèrent la formule d'entrée sous forme prénexe et clausale et sont des variantes de la fameuse procédure de Davis, Logemann and Loveland (DLL) [8]. Cependant, un des gros inconvénients de ces approches vient du fait que les choix heuristiques opérés aux différents noeuds de l'arbre de recherche doivent respecter l'ordre du préfixe (du quantificateur le plus externe au plus interne). Cet ordre statique limite l'efficacité de ces solveurs QBF basés sur la recherche systématique. En effet, dans certains cas, la difficulté des instances QBF peut être due aux variables apparaissant dans les quantificateurs les plus internes.

L'objectif principal de cet article est de rendre les sol-

veurs libres de cet ordre imposé par les quantificateurs et de faciliter l'extension des techniques efficaces couramment utilisées par les solveurs SAT. A cette fin, les diagrammes de décision binaires (BDD) vont être utilisées pour représenter dans une forme compacte l'ensemble des modèles trouvés sur la partie clausale de la formule. Ainsi, nous proposons un nouveau type de solveur QBF, nommé QBDD(SAT), combinant les techniques de recherche pour le problème de satisfaisabilité avec les diagrammes de décision binaires. Pour des raisons de complétude et d'efficacité, deux points clés sont ajoutés à notre approche. D'une part, nous introduisons un nouvel opérateur de réduction pour les BDD. Il va permettre de réduire la taille de ces derniers et de répondre à la validité des formules QBF. De l'autre, pour chaque modèle propositionnel trouvé, seul un impliquant premier de celui ci va être encodé dans le BDD et un «*nogood*» va être rajoutée à la formule. La coupure engendrée par ces *nogoods* supprimera certains modèles inutiles.

Cet article étend des résultats préliminaires [2]. De nouvelles fonctionnalités ont été rajoutés et deux techniques de résolution du problème SAT sont étendus en utilisant notre approche QBDD(SAT), d'un coté les solveurs de type DLL et de l'autre, les solveurs de recherche locale.

Le reste du papier est organisé comme suit. Après les définitions d'usage (problème QBF et diagramme de décision binaire), nous montrons comment les diagrammes de décision binaires peuvent naturellement être intégrés aux techniques de résolution du problème SAT pour pouvoir résoudre les QBF. Nous présentons ensuite nos deux solveurs. Le premier QBDD(DLL) est basé sur la procédure DLL, le second est lui basé sur la recherche locale. Nous terminons par des expérimentations et une conclusion.

2 Définitions

Dans cette section, nous rappelons brièvement les notations et définitions nécessaires sur les formules booléennes quantifiées et sur les diagrammes de décision binaires.

2.1 Formules Booléennes Quantifiées

Soit $\mathcal{P}$ un ensemble de variables propositionnelles. $\mathcal{L}_\mathcal{P}$ désigne alors le langage des formules booléennes quantifiées construites à partir de $\mathcal{P}$ en utilisant les formules booléennes classiques (incluant les constantes $\top$ et $\bot$) ainsi que les quantifications $\exists$ et $\forall$.

On considère les formules booléennes quantifiées mises sous forme prénexe : $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ (ou encore $QX\Psi$, QX est le préfixe de Φ et Ψ sa matrice) tel que $Q_i \in \{\exists, \forall\}$, $X_k, \ldots, X_1$ sont des ensembles disjoints de variables et Ψ une formule booléenne. Les variables consécutives ayant le même quantificateur sont groupées. Le rang d'une variable $x \in X_i$ est égal à i. Les variables apparaissant dans le même groupe de quantificateurs ont donc le même rang. L'ordre induit par le préfixe de la formule QBF $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ est obtenu en prenant en compte le rang des variables et est noté $X_k < X_{k-1} < \cdots < X_1$. Une QBF Φ est dite sous forme normale si Φ est sous forme prénexe et si Ψ est sous forme normale conjonctive (CNF). On peut noter que le quantificateur le plus interne peut être considéré comme existentiel. En effet, lorsque Q_1 est universel, la suppression de $\forall X_1$ du préfixe et de toutes les occurrences de $x \in X_1$ dans la matrice conduit à une formule équivalente.

Afin de définir la sémantique des formules booléennes quantifiées, nous introduisons quelques notations. Soit S l'ensemble des affectations possibles basées sur l'ensemble des variables V. La Uprojection (resp. la Dprojectionn) d'un ensemble d'affectation S sur l'ensemble des variables de $X \subset V$, notée $S \uparrow X$ (resp. $S \downarrow X$), est obtenue en restreignant chaque affectation de S aux littéraux de X (resp. dans $V \backslash X$). L'ensemble des affectations possibles sur X est notée 2^X. Une affectation sur X est notée par le vecteur $\overrightarrow{x}$. De même, Uprojection et Dprojectionn peuvent également s'appliquer sur un vecteur de littéraux $\overrightarrow{x}$. Si $\overrightarrow{y}$ est une affectation sur Y tel que $Y \cap X = \emptyset$, alors $\overrightarrow{y}.S$ désigne l'ensemble des interprétations obtenues en concaténant $\overrightarrow{y}$ avec chaque interprétation de S. Finalement, $\Psi(\overrightarrow{x})$ désigne la formule Ψ simplifiée par l'affectation partielle $\overrightarrow{x}$.

Une QBF est valide (vraie) s'il existe une solution, appelée *politique totale* définie comme suit. Cette définition est une version simplifiée de celle apparaissant dans [7].

Définition 1 *Soient $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ une formule booléenne quantifiée et $\pi = \{m_1, \ldots, m_n\}$ un ensemble de modèles de la formule booléenne Ψ. π est une politique totale de la QBF Φ si et seulement si π vérifie récursivement les conditions suivantes :*

1. *$k = 0$, et $\Psi = \top$*
2. *Si $Q_k = \forall$, alors $\pi \uparrow X_k = 2^{X_k}$, et $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ est une politique totale de $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$*
3. *Si $Q_k = \exists$, alors $\pi \uparrow \overrightarrow{X}_k = \{\overrightarrow{x}_k\}$ et $\pi \downarrow \overrightarrow{x}_k$ est une politique totale de $Q_{k-1} X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$*

Remarque 1 *Soit π une politique totale de $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$. Si $Q_k = \forall$ alors on peut réécrire π de la manière suivante $\bigcup_{\overrightarrow{x}_k \in 2^{X_k}} \{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$ et si $Q_k = \exists$, alors $\pi \uparrow X_k = \{\overrightarrow{x}_k\}$ et π peut être réécrit comme $\{\overrightarrow{x}_k.(\pi \downarrow \overrightarrow{x}_k)\}$*

Exemple 1 *Soit $\Phi = \exists x_5 x_6 \forall x_2 x_4 \exists x_1 x_3 \Psi$ une formule QBF, où $\Psi = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_4 \vee x_3) \wedge (x_5 \vee x_6)$. Φ est valide et $\pi = \{(\neg x_5, x_6, x_2, x_4, \neg x_1, x_3), (\neg x_5, x_6, x_2, \neg x_4, \neg x_1, x_3), (\neg x_5, x_6, \neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_5, x_6, \neg x_2, x_4, x_1, x_3)\}$*

est une politique totale de Φ (illustré dans figure 1). Les différents opérateurs de projection sont illustrés ci dessous : $\pi \uparrow \{x_5, x_6\} = \{(\neg x_5, x_6)\}$

$\pi' = \pi \downarrow \{x_5, x_6\}$
$= \{(x_2, x_4, \neg x_1, x_3), (x_2, \neg x_4, \neg x_1, x_3),$
$(\neg x_2, \neg x_4, x_1, \neg x_3), (\neg x_2, x_4, x_1, x_3)\}$

$\pi' \uparrow \{x_2, x_4\} = \{(\neg x_2, \neg x_4), (\neg x_2, x_4), (x_2, \neg x_4),$
$(x_2, x_4)\}$
$= 2^{\{x_2, x_4\}}$

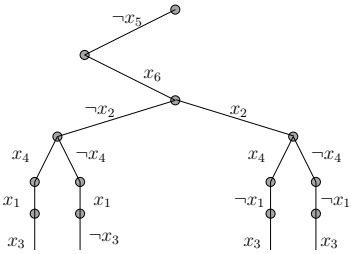


FIG. 1 – Représentation arborescente d'une politique totale (exemple 1)

Motivé par les résultats impressionnants obtenus dans la résolution du problème de satisfaisabilité, plusieurs solveurs QBF ont été proposés récemment. La plupart d'entre eux sont des extensions de la fameuse procédure DLL et possèdent les dernières techniques efficaces connues pour résoudre le problème SAT, comme l'apprentissage, les heuristiques et la propagation des contraintes. Par exemple, QUBE [12] et QUAFFLE [18] utilisent le «*backjumping*» et l'apprentissage, EVAUATE[6] et DECIDE [16] étendent les techniques de coupure de l'arbre de recherche existant pour le problème SAT (propagation unitaire, litéraux purs...)

L'algorithme 1 donne le schéma général d'une procédure DLL basique utilisée pour répondre à la validité des formules QBF. Elle prend en entrée les variables du préfixe $< X_k, \ldots, X_1 >$ associé aux quantificateurs $Q_k, \ldots, Q_1$ et la matrice Ψ sous forme clausale. Elle retourne `true` si la formule QBF $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ est valide, `false` autrement. L'algorithme commence par simplifier la formule en propageant les litéraux unitaires et purs (procédure `Simplify`). Ensuite, si la matrice simplifiée contient une clause vide, alors la formule courante est invalide (la valeur `false` est retournée), autrement si la matrice courante est vide alors la formule QBF est valide (la valeur `true` est retournée). L'étape suivante consiste à choisir la prochaine variable à instancier, variable qui doit appartenir au groupe de quantificateur (non vide) X_k le plus externe. Ceci diffère de la procédure DLL pour le problème SAT, puisque les variables doivent donc être instanciées suivant l'ordre associé au préfixe.

Dépendant du quantificateur Q_k de la variable choisie, une branche de gauche et/ou une branche de droite dans l'arbre de recherche va être générée. Si $Q_k = \forall$ (resp. $Q_k = \exists$) la branche de droite est générée seulement si la valeur retournée par la branche de gauche est la valeur `true` (resp. `false`).

Algorithm 1: DLL for QBF

Data : Ψ : matrice de la QBF ; $< X_k, \ldots, X_1 >$: préfixe de la QBF Φ
Result : true si la QBF Φ est valide, faux autrement
begin
  Simplify(Ψ);
  if $\emptyset \in \Psi$ **then return** false;
  if $\Psi = \emptyset$ **then return** true;
  if $X_k = \emptyset$ **then**
    return QDLL($\Psi, < X_{k-1}, \ldots, X_1 >$);
  choisir (par heuristique) un litéral $l \in X_k$;
  if *(($Q_k = \forall$) and QDLL($\Psi \cup \{l\}, < X_k - \{l\}, \ldots, X_1 >$)=false)*
  then
    return false;
  if *(($Q_k = \exists$) and QDLL($\Psi \cup \{l\}, < X_k - \{l\}, \ldots, X_1 >$)=true)*
  then
    return true;
  return
  QDLL($\Psi \cup \{\neg l\}, < X_k - \{l\}, \ldots, X_1 >$);
end

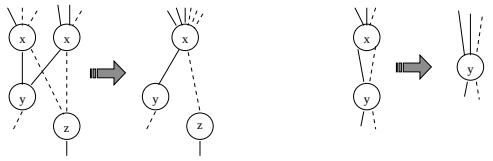
L'ordre d'affectation imposé par le préfixe constitue l'un des gros désavantages de l'extension de la procédure DLL pour les QBF. Les nombreuses recherches (voir par exemple [9, 14]) sur les heuristiques de la procédure DLL pour SAT montre que le choix des variables est très important. L'ordre imposé par le préfixe peut donc avoir de grosses conséquences sur les performances des solveurs QBF. De plus, cette limitation peut rendre difficile certaines extensions de techniques issues du problème de satisfaisabilité (voir par exemple [9] pour les problèmes aléatoires ou encore [14] pour les problèmes structurés). Un autre exemple vient de la recherche locale [17], autre paradigme couramment utilisé pour résoudre le problème SAT. Ce type de recherche a reçu beaucoup moins d'attention au niveau QBF. Une première intégration de la recherche locale dans les solveurs QBF (`WalkQSAT`) a été proposé dans [11]. Mais, `WalkQSAT` est une implémentation de recherche systématique dirigée par conflit et par «*backjumping*» pour les QBF. Ce solveur utilise uniquement la recherche locale pour guider sa recherche.

2.2 Diagrammes de décision binaires

Un diagramme de décision binaire (BDD) [1, 5] est associé à une formule booléenne. C'est un graphe acyclique avec une racine et deux noeuds terminaux notés 1-terminal et 0-terminal. Tout noeud non terminal est associé à une

variable de la formule et possède deux arcs sortant, appelés arc-1 correspondant à la variable assignée à vrai, et arc-0 correspondant à la variable assignée à faux. Un diagramme de décision binaire ordonné (OBDD) est un BDD tel que les variables apparaissent au plus une fois et dans un ordre fixe dans tout chemin du graphe. Un OBDD réduit (ROBDD) est un OBDD résultant de l'application répétée des règles suivantes :

1. Élimination des sous graphes isomorphes (figure 2.a).
2. Élimination des noeuds pour lesquels les deux arcs sortants pointent sur le même noeud (figure 2.b).



a. Élimination des sous graphes isomorphes

b. Élimination de noeud redondant

FIG. 2 – Règles de réduction

Un ROBDD représentant la formule booléenne Ψ est noté $(RO)BDD(\Psi)$.

La figure 3 illustre la représentation en ROBDD de la politique π ($ROBDD(\pi)$) de l'exemple 1. Les arcs solides représentent les arcs-1 et les arcs en pointillé représentent les arcs-0. L'ordre du ROBDD est le même que l'ordre du préfixe de la formule Φ, à savoir ($\{x_5, x_6\} < \{x_2, x_4\} < \{x_1, x_3\}$)

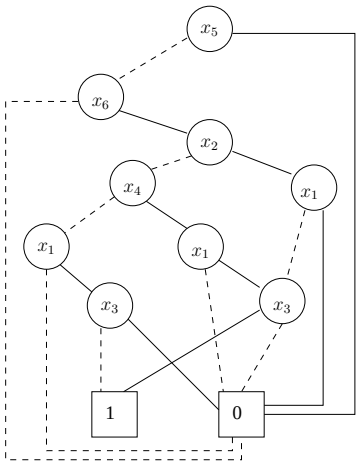


FIG. 3 – Représentation d'une politique sous forme de BDD (exemple 1)

Les ROBDDs ont des propriétés très intéressantes. Il donnent une représentation compacte et unique (par rapport à un ordre donné) de fonctions booléennes. De plus, à partir des ROBDD, il existe des algorithmes efficaces pour réaliser diverses opérations logiques. Par exemple, il est possible de savoir si une formule booléenne est satisfaisable en temps constant. Malgré leur taille exponentielle dans le pire des cas, les ROBDD font parties des structures de données les plus utilisées dans la pratique.

Dans le reste de l'article, seuls les ROBDD sont considérés et, par simplification, nous les appellerons BDD. Lorsque les formules booléennes quantifiées seront considérées, l'ordre du BDD suivra l'ordre du préfixe de la formule.

3 QBDD(SAT) : Une approche de recherche symbolique

Afin de libérer les solveurs QBF de l'ordre induit par le préfixe, nous proposons une combinaison des solveurs SAT classiques et des diagrammes de décision binaires. La figure 4 donne l'architecture générale de notre approche symbolique QBDD(SAT). Plus précisément, pour vérifier la validité d'une formule QBF $\Phi = QX\Psi$, notre approche utilise les techniques de résolution du problème SAT pour rechercher les modèles de la formule booléenne Ψ (*SAT enumerator*). Pour chaque modèle m trouvé, un impliquant premier est extrait (*Compute PI*) et est ajouté disjonctivement au BDD (*bdd = or(bdd, pi)*), BDD dont l'ordre est en accord avec l'ordre des variables dans le préfixe. Lorsque l'ensemble courant des impliquants premiers représente une politique totale, sa représentation sous forme de BDD sera réduit au noeud 1-terminal (voir section 3.1) et le solveur QBDD(SAT) répondra donc à la validité de Φ (valeur `valid`). Comme nous l'avons précédemment mentionné, l'approche QBDD(SAT) peut être instanciée avec différentes techniques de recherche pour le problème SAT. Par exemple, QBDD(DLL) (resp QBDD(LS)) se réfère à un solveur QBF obtenu en prenant comme solveur SAT une procédure de type DLL (resp. recherche locale). A la fin du processus de recherche propositionnelle, si le BDD n'est pas réduit au noeud 1-terminal, alors suivant la complétude du solveur SAT utilisé, notre approche QBDD(SAT) permettra ou non de dire si la formule QBF est invalide.

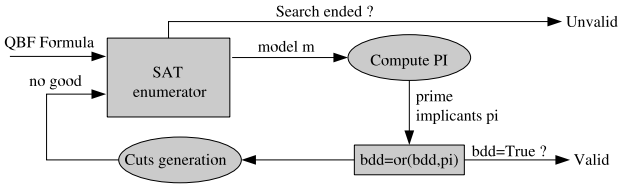


FIG. 4 – architecture de QBDD(SAT)

Pour des raisons de complexité, seuls les impliquants de la formule sont encodés dans les BDD, les «*nogood*» trouvés durant la phase de recherche propositionnelle ne sont pas considérés. Bien entendu, les chemins qui mènent au

noeud 0-terminal ne représentent pas des contre modèles de la formule Ψ. Ainsi, tous les chemins menant à ce noeud peuvent être omis. Enfin, pour permettre de réduire l'espace de recherche, un *nogood* est généré à partir de chaque modèle propositionnel et du BDD (*Cuts Generation*). Il permet d'éviter de rechercher des modèles inutiles.

3.1 Opérateurs de réductions dus aux quantificateurs

Afin de réduire la taille des BDD et de répondre à la question de la validité d'une formule QBF, nous introduisons un nouvel opérateur de réduction sur les BDD. Cet opérateur est montré dans la figure 5. Lorsque un noeud x est quantifié existentiellement et que l'un de ses arcs sortants référence le noeud 1-terminal alors toute référence à ce noeud x peut être simplement remplacé par une référence au noeud 1-terminal. Cette règle est appelée opérateur de réduction existentielle. Rappelons de plus que lorsque un noeud x est quantifié universellement et que ses deux arcs sortants référencent le noeud 1-terminal, le noeud x peut être éliminé par un opérateur de réduction classique (figure 2.b).

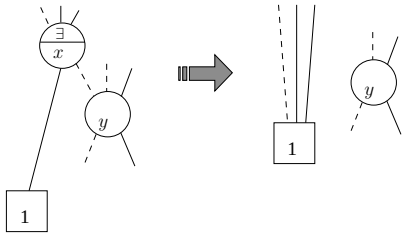


FIG. 5 – Réduction existentielle

Durant la construction du BDD, nous ajoutons l'opérateur de réduction existentielle aux opérateurs de réduction classiques (figure 2). La propriété suivante nous assure qu'un ensemble de modèles est une politique totale si la représentation de cette dernière dans le BDD est réduite au noeud 1-terminal.

Propriété 1 *Soient $\Phi = Q_k X_k, \ldots, Q_1 X_1 \Psi$ une formule booléenne quantifiée et $\pi = \{\overrightarrow{x}_1, \overrightarrow{x}_2, \ldots, \overrightarrow{x}_n\}$ un ensemble de modèle de Ψ. Si π est une politique totale de Φ alors BDD(π) et réduit au noeud 1-terminal.*

Preuve : La preuve est obtenu par induction sur k. Si $k = 0$, la représentation de la constante $\top$ est le noeud 1-terminal (par définition d'une politique totale). Supposons maintenant que la propriété est vrai pour $k-1$, prouvons qu'elle est vraie pour k. Par définition d'une politique totale, deux cas peuvent se produire :

1. si $Q_k = \forall$, alors $\pi \uparrow X_k = 2^{X_k}$, et $\forall \overrightarrow{x}_k \in 2^{X_k}$, $\pi \downarrow \overrightarrow{x}_k$ est une politique totale de la formule QBF $Q_{k-1}X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$. Par hypothèse d'induction, on peut déduire que BDD($\pi \downarrow \overrightarrow{x}_k$) est réduit au noeud 1-terminal. Donc, toutes les feuilles de BDD(2^{X_k}) sont des noeuds 1-terminaux. En appliquant répétitivement la règle d'élimination des noeuds redondants (figure 2.a) sur le BDD résultant, on obtient un BDD réduit au noeud 1-terminal.
2. Si $Q_k = \exists$, alors $\pi \uparrow \overrightarrow{X}_k = \{\overrightarrow{x}_k\}$ et $\pi \downarrow \overrightarrow{x}_k$ est une politique totale de la formule QBF $Q_{k-1}X_{k-1}, \ldots, Q_1 X_1 \Psi(\overrightarrow{x}_k)$. Par hypothèse d'induction BDD($\pi \downarrow \overrightarrow{x}_k$) est un noeud 1-terminal. Donc le BDD($\{\overrightarrow{x}_k\}$) peut être vu comme une branche allant au noeud 1-terminal. En appliquant répétitivement la règle de réduction existentielle, le BDD(π) se réduit au noeud 1-terminal.

Remarque 2 *Dualement, l'opérateur de réduction universelle peut aussi être défini. Mais comme le noeud 0-terminal représente un état indéfini, il ne peut pas être utilisé dans notre approche.*

L'exemple suivant montre les différentes réductions qui vont être effectuées sur l'ensemble des modèles représentant la politique totale de l'exemple 1.

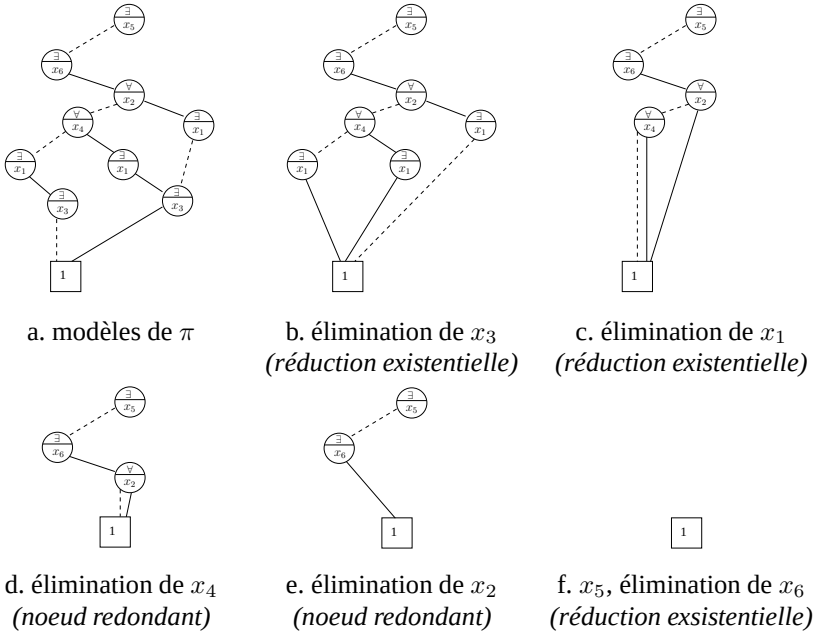
Exemple 2 *Soient Φ la formule QBF de l'exemple 1 et π la politique totale de cette dernière. La figure 6 représente les différentes étapes de réduction opérées sur BDD(π) :*

- *La figure 6.a est la représentation en BDD de la politique π (seuls les chemins menant au noeud 1-terminal sont représentés).*
- *L'opérateur de réduction existentielle permet l'élimination du noeud x_3 (figure 6.b) et du noeud x_1 (figure 6.c).*
- *L'opérateur de réduction des noeuds redondants permet l'élimination de x_4 (figure 6.d) et de x_2 (figure 6.e).*
- *Enfin, l'opérateur de réduction existentielle supprime les noeuds x_5 et x_6. Le BDD se retrouve réduit au noeud 1-terminal représentant la formule vrai (figure 6.f) et prouvant que π est une politique.*

3.2 Génération de coupures grâce au BDD

Pour éviter de rechercher des modèles qui appartiennent à des mêmes politiques potentielles, nous allons générer des coupures dans l'arbre de recherche. Ces coupures sont réalisées grâce à des *nogoods* injectés dans la formule de départ. Les *nogoods* sont construits à partir des modèles obtenus et du BDD. Il est important de noter que ces *nogoods* n'ont rien de commun avec les *nogoods* obtenus par apprentissage dans les solveurs de type `zchaff`.

Définition 2 *Soient $\Phi = Q_k X_k, \ldots, Q_2 X_2, Q_1 X_1 \Psi$ une QBF telle que $Q_2 = \forall$, $Q_1 = \exists$ et $\overrightarrow{x}$ un modèle de*



a. modèles de π | b. élimination de x_3 *(réduction existentielle)* | c. élimination de x_1 *(réduction existentielle)*

d. élimination de x_4 *(noeud redondant)* | e. élimination de x_2 *(noeud redondant)* | f. x_5, élimination de x_6 *(réduction exsistentielle)*

FIG. 6 – Phase de réduction du BDD d'une politique totale

Ψ. *On définit* $nogood_m(\overrightarrow{x}) = \bigvee\{\neg l | l \in \overrightarrow{x} \downarrow X_1\}$ *comme le nogood obtenu à partir de* $\overrightarrow{x}$. *De même, on définit* $nogood_{pi}(\overrightarrow{pi})$ *comme le nogood extrait de l'impliquant premier de* $\overrightarrow{pi}$.

Naturellement, si $\overrightarrow{pi}$ est un impliquant premier obtenu à partir du modèle $\overrightarrow{x}$ alors $nogood_{pi}(\overrightarrow{pi}) \vDash nogood_m(\overrightarrow{x})$. En utilisant l'exemple 1, la figure 7, montre que pour le modèle $\overrightarrow{x} = \{\neg x_5, x_6, \neg x_2, x_4, x_1, x_3\}$, le $nogood_m(\overrightarrow{x}) = (x_5 \vee \neg x_6 \vee x_2 \vee \neg x_4)$ évite de rechercher des modèles qui appartiennent aux mêmes politiques potentielles. En considérant l'impliquant premier $\overrightarrow{pi} = \{x_6, x_1, x_3\}$ de $\overrightarrow{x}$, on peut générer une coupure plus importante, en effet $nogood_{pi}(\overrightarrow{pi}) = (\neg x_6)$.

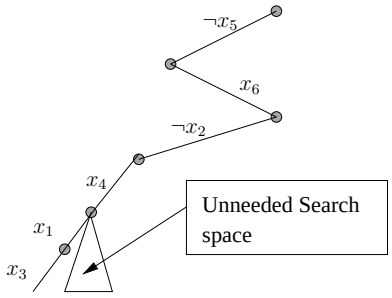


FIG. 7 – Génération de coupures

Rappelons que lorsque on ajoute disjonctivement un impliquant premier au BDD, les opérateurs de réduction vont éliminer toutes les variables de X_1. De plus, les variables de X_2 peuvent également être éliminées si les deux arcs sortants référencent le noeud 1-terminal. Ce processus est répété récursivement. Ainsi, pour générer des coupures plus importantes, à chaque fois qu'un impliquant premier est ajouté au BDD, le *nogood* est extrait du nouveau BDD réduit et est ensuite ajouté à la formule propositionnelle.

3.3 L'approche QBDD(DLL)

L'algorithme 2 représente le solveur QBF obtenu en instanciant notre approche avec une procédure de type DLL. L'algorithme résultant recherche tous les modèles de la formule propositionnelle jusqu'à ce que le BDD soit réduit au noeud 1-terminal, dans ce cas l'algorithme termine et retourne *valid*. Si l'algorithme retourne au niveau 0, tous les modèles nécessaires ont été trouvé mais ne forment jamais de politique. Dans ce cas l'algorithme retourne *unvalid*. Dans tous les autres cas, la recherche continue (la valeur *back* est retournée). La fonction $Simplify()$ simplifie la formule Ψ (propagation unitaire), alors que la fonction $conflictAnalysis()$ implémente la technique d'apprentissage utilisée dans de nombreux solveurs. La fonction $primeImpliquants()$ extrait l'impliquant premier du modèle I obtenu. Enfin, la fonction $cutsGeneration()$ génère, à partir du BDD courant, le nouveau *nogood* et le rajoute à la formule booléenne Ψ (voir la section 3.2). L'exemple 3 donne une trace possible de cet algorithme.

Exemple 3 *Considérons la formule* Φ *de l'exemple 1. Supposons que l'algorithme* QBDD(DLL) *commence la recherche par assigner* x_1 *et* x_5. *A ce stade, on doit satisfaire la clause* $(\neg x_4 \vee x_3)$. *Supposons maintenant que le choix de la prochaine variable de branchement se porte sur* $\neg x_4$. *Un*

premier modèle est obtenu $(x_5, \neg x_4, x_1)$ *et est ajouté au bdd. Ce dernier est donc réduit à un seul chemin jusqu'au noeud 1-terminal :* $< x_5, \neg x_4 >$ *(la variable* x_1 *est supprimée par réduction existentielle). La clause* $(\neg x_5 \vee x_4)$ *est rajoutée à la formule* Ψ*. Un retour arrière est opéré pour chercher d'autres modèles et l'assignation de* x_4 *implique celle de* x_3 *par propagation unitaire. Un second modèle est obtenu* (x_5, x_4, x_1, x_3) *et est rajoutée disjonctivement au bdd qui se retrouve réduit au noeud 1-terminal par réduction existentielle et des noeuds redondants. La recherche s'arrête donc, montrant la validité de la formule* Φ*.*

Algorithm 2: Le solveur QBDD(DLL) : combinaison de BDD et de DLL

Data : Ψ : ensemble de clauses ; X=$\{X_k, \ldots, X_1\}$: préfixe de la QBF; I : interprétation partielle ; d : niveau de recherche dans l'arbre, initialement vaut 0.
Result : $valid$ si la QBF est valide, $unvalid$ sinon; $back$ est retournée si la recherche doit continuer pour trouver un autre modèle propositionnel.

```
begin
    Simplify(Ψ);
    if ∅ ∈ Ψ then
        conflictAnalysis();
        return back;
    if Ψ = ∅ then
        pi := primeImpliquants(I, Ψ);
        bdd := or(bdd, pi);
        if bdd=1-terminal) then return valid;
        cutsGeneration(pi,bdd);
        return back;
    Soit l ∈ X_i (i ∈ {1...k}) la prochaine variable
    de branchement;
    if (QBDD(DLL)
    (Ψ ∪ {l}, X − {l}, I ∪ {l}, d + 1)=valid) or
    QBDD(DLL)
    (Ψ ∪ {¬l}, X − {l}, I ∪ {¬l}, d + 1)=valid)
    then
        return valid;
    if (d = 0) then
        return unvalid;
    return back;
end
```

3.4 Le solveur QBDD(LS)

L'une des fonctionnalités intéressantes de notre approche QBDD(SAT) vient du fait que que l'on peut utiliser assez facilement n'importe quelle algorithme de recherche pour le problème de satisfaisabilité. Particulièrement, les techniques de recherche locale peuvent aisément s'intégrer. En utilisant le solveur `WalkSAT` nous obtenons un nouveau solveur QBDD(LS). Le résultat est un solveur incomplet qui ne peut prouver que la validité des formules. Ce solveur diffère de `WalkQSat` [11]. En effet, QBDD(LS) utilise la recherche locale comme un générateur de modèles et `WalkQSat` l'utilise pour guider la procédure de recherche DLL.

4 Évaluation empirique

Les résultats expérimentaux reportés dans cette section sont obtenus sur un Pentium IV 3 GHz avec 1GB RAM et sont réalisés sur un large panel d'instances (644 instances) issues de l'évaluation QBF03 [3]. Ces instances sont divisées en diverses familles (`log`, `impl`, `toilet`, `k_*`...). Pour chaque exécution, le temps, exprimé en secondes, est limité à 600 secondes. Le solveur QBDD(DLL) est implémenté à partir de `minisat` [10], un solveur à la `zchaff`. Quant à QBDD(LS) il est bâti sur `WalkSAT`.

4.1 Comportement du solveur QBDD(DLL)

La figure 8 montre le comportement des différentes versions du solveur QBDD(DLL) :

- QBDD(DLL) est la version basique sans calcul d'impliquants premiers ni la génération de coupures
- QBDD(DLL) +CUTS est augmenté de la génération de coupures
- QBDD(DLL) +PI calcule les impliquants premiers de chaque modèle
- QBDD(DLL) +PI+CUTS contient ces deux fonctionnalités

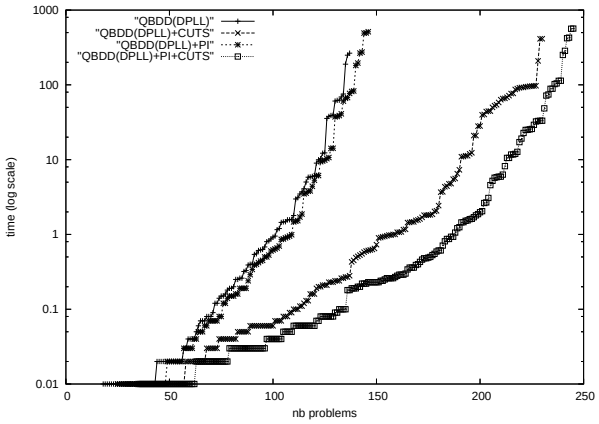


FIG. 8 – Nombre d'instances résolues vs tempsCPU

L'axe des x de la figure 8 représente le nombre de benchmarks résolus et l'axe des y (échelle logarithmique) le temps nécessaire pour résoudre le nombre de problèmes donné. La figure 8 montre clairement que la version basique est la moins efficace. Si on n'ajoute que la fonctionnalité des impliquants premiers, on n'améliore que très peu

problème	valide	QBDD(DLL)	QBDD(DLL) +PI	QBDD(DLL) +CUTS	QBDD(DLL) +PI+CUTS
impl06	Y	6112	6012	2142	550
k_poly_n-1	Y	>45000	>45000	1850	862
toilet_a_10_01.5	N	41412	41412	22486	20 917

TAB. 1 – Nombre de modèles

la version basique. Par contre, la génération des coupures dans l'arbre (par rajout de *nogood*) permet une réelle amélioration, le nombre de problèmes résolus en moins de 600 secondes passant de 130 à 220. Finalement, la meilleure version du solveur QBDD(DLL) est obtenu en ajoutant les coupures et en calculant les impliquants premiers.

La table 1 donne quelques explications sur ce résultat. Le nombre de modèles nécessaires pour répondre à la question de la validité y est reporté. Comme seuls les modèles nécessaires sont calculés lorsque l'on rajoute des coupures dans l'arbre de recherche, le nombre de modèles calculés est beaucoup plus petit avec cette fonctionnalité que sans. La génération d'impliquants premiers demande encore moins de modèles puisque ceux ci englobent une plus grande portion de politiques potentielles.

La table 2 montre le nombre d'instances résolus par rapport aux nombre de quantificateurs. Notre approche semble vraiment efficace avec les 2QBF et beaucoup moins avec des préfixes de grande taille (seuls 11.6% des problèmes avec une taille du préfixe supérieure à 8 sont résolus). Pour autant, QBDD(DLL) n'est pas réduit aux 2QBF puisque il permet de résoudre la moitié des 5QBF. Ces résultats sont en accord avec ceux obtenus dans [15] sur des 2QBF en utilisant une combinaison originale de deux solveurs SAT.

taille du préfixe	2	3	4	5	6	7	≥ 8
nb problèmes	57	339	8	34	7	22	154
nb résolus	51	148	1	19	1	11	18
%	89.5	43.6	12.5	55.8	14.2	50	11.6

TAB. 2 – Instances résolues et taille du préfixe

4.2 Comparaison avec différents solveurs

Nous comparons le solveur QBDD(DLL) (exploitant le calcul d'impliquants premiers et la génération de coupures) avec QUBE [12] et QUANTOR [4], deux solveurs parmi les plus efficaces. La table 3 montre une comparaison sur différentes familles d'instances. La colonne *#N* désigne le nombre d'instances dans une famille donnée, la colonne *#S* désigne le nombre d'instances résolus par un solveur donné, et *TT*, le temps total nécessaire au solveur pour résoudre toutes les instances de la famille. Lorsque un solveur ne résout pas une instance, 600 secondes sont rajoutées au temps total.

Les moins bons résultats de QBDD(DLL) sont obtenus sur les instances de la famille `toilet`. Seules 106 instances sont résolues en moins de 600 secondes, alors que QUBE et QUANTOR les résolvent assez facilement. C'est la même chose pour les instances de la famille `k_*`. Les meilleurs résultats sont obtenus sur la famille `robot`. Plus d'instances y sont résolues et plus rapidement que les deux autres solveurs. De plus, certaines instances sont résolues pour la première fois. En effet, les instances `robot` se composent deux groupes de quantificateurs et contiennent beaucoup de variables universelles (et donc beaucoup de modèles propositionnels sont nécessaires pour prouver leur validité). les clauses contenant les variables apparaissant dans le groupe de quantificateur interne constituent la partie difficile. Un solveur comme QUBE doit donc résoudre de nombreuses fois cette partie. Enfin, Sur un large panel de familles (`z4`, `flipflop`, `log`), les résultats des trois solveurs ont des résultats comparables.

		Qube		Quantor		QBFBDD	
family	#N	#S	TT	#S	TT	#S	TT
robots	48	36	7 214	35	7 970	42	4 270
k_*	171	98	44 813	99	43 786	32	83 658
flipflop	7	7	0.36	7	1.2	7	3.2
toilet	260	260	40	260	256	106	93 860
impl	8	8	0.01	8	0.02	6	1211
tree-exa10	6	6	0.07	6	0.01	6	1.7
chain	8	8	497	8	0.3	2	4183
tree-exa2	6	6	0.01	6	0.01	1	3000.1
carry	2	2	0.23	2	0.69	2	0.71
z4	13	13	0.06	13	0.08	13	0.1
blocks	3	3	0.2	3	0.47	3	3.2
log	2	2	18.4	2	42	2	31

TAB. 3 – Comparaison entre Qube, Quantor et QBDD(DLL)

4.3 Comparaison avec QBDD(LS)

Comme QBDD(LS) est un solveur incomplet qui ne peut prouver que la validité des instances, il est difficile de faire une comparaison juste avec d'autres solveurs. Néanmoins, nous présentons les premiers résultats obtenus sur des instances valides et faisons une comparaison avec QBDD(DLL), `Quantor` et `Qube`. La table 4 donne le temps et le nombre de modèles calculés durant la recherche (si celui ci est donné). Pour QBDD(LS), chaque instance est résolue 20 fois et la médiane est reportée.

Ces résultats expérimentaux montre que l'approche QBDD(LS) est prometteuse. Elle résoud 76 des 150 instances valides de l'évaluation QBF03. Elle est assez compétitive et obtient quelques fois des résultats meilleurs que les autres solveurs.

	QBDD(LS)		QBDD(DLL)		Quantor	Qube
problem	model	time	model	time	time	time
impl10	3440	8	2673	2	0.01	0.01
toilet_c_04_10.2	2850	14	?	>600	0.01	0.01
robots_1_5_2_3.3	58	6	79	0.28	453	0.7
comp.blif_0.10_1.00_0_1_out_exact	3205	308	4647	1.8	0.03	>600
tree_exa10-20	?	>600	1095	0.2	0.01	0.1

TAB. 4 – Comparaison sur des instances valides

5 Conclusion

Dans cet article, nous présentons une nouvelle approche symbolique pour résoudre les formules QBF. Cette approche consiste en une combinaison originale des techniques de recherche de modèles propositionnels et des diagrammes de décision binaires. Les BDD sont utilisés pour encoder les impliquants premiers de la formule booléenne. Un nouvel opérateur de réduction des BDD est proposé. Il permet de réduire la taille de ce dernier et de répondre à la question de la validité des QBF. Des *nogoods* sont calculés à partir du BDD et permettent d'opérer de sérieuses coupes dans l'espace de recherche. L'avantage principal de cette approche est qu'elle permet de s'abstenir de l'ordre des quantificateurs. Cela facilite l'extension des solveurs SAT pour résoudre des QBF. Deux paradigmes de recherche ont été proposé (recherche systématique et recherche stochastique), donnant lieu à deux solveurs (QBDD(DLL) et QBDD(LS)). Les résultats expérimentaux ont montré l'efficacité de notre approche. Plus intéressant, certaines instances difficiles ont été résolus pour la première fois.

Références

[1] S. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27 :509–516, 1978.

[2] Gilles Audemard and Lakhdar Sais. Sat based bdd solver for quantified boolean formulas. In *proceedings of the 16th IEEE international conference on Tools with Artificial Intelligence*, pages 82–89, 2004.

[3] Daniel Le Berre, Laurent Simon, and Armando Tachella. Challenges in the qbf arena : the sat'03 evaluation of qbf solvers. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT2003)*, *LNAI*, pages 452–467, 2003.

[4] Armin Biere. Resolve and expand. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2004.

[5] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 8 :677–692, C-35.

[6] Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. An algorithm to evaluate quantified boolean formulae. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 262–267, Madison (Wisconsin - USA), 1998.

[7] Sylvie Coste Marquis, Helene Fargier, Jerome Lang, Daniel Le Berre, and Pierre Marquis. Résolution de formules booléennes quantifiées : problèmes et algorithmes. In *Actes du 13eme congrès AFRIF-AFIA Reconnaissance des forme et intelligence artificielle (RFIA)*, pages 289–298, 2002.

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

[9] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI–01)*, August 4–10 2001.

[10] Niklas Eén and Niklas Sörensson. An extensible sat solver. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, p. 502–508, 2003.

[11] Ian P. Gent, Holger H. Hoos, Andrew G. D. Rowley, and Kevin Smyth. Using stochastic local search to solve quantified boolean formulae. In *Proceedings of the 9th international conference of principles and practice of constraint programming*, pages 348–362, 2003.

[12] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. QuBE : A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, Italy, June 2001.

[13] Reinhold Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of Tableaux 2002*, pages 160–175, Copenhagen, Denmark, 2002.

[14] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engi-

neering an efficient sat solver. In *Proceedings of 38th Design Automation Conference (DAC01)*, 2001.

[15] Darsh Ranjan, Daijue Tang and Sharad Malik Niklas. A Comparative Study of 2QBF Algorithms. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[16] Jussi Rintanen. Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae. In *Proceedings of the First International Conference on Quantified Boolean Formulae (QBF'01)*, pages 84–93, 2001.

[17] Bart Selman, Hector Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 459–465, 1992.

[18] Lintao Zhang and Sharad Malik. Towards a symetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, pages 200–215, 2002.