Calcul propositionnel : Vers une extension du formalisme

A. Rauzy
LaBRI – CNRS – Université Bordeaux I
351, cours de la Libération
33405 Talence Cedex – FRANCE
rauzy@labri.u-bordeaux.fr

L. Saïs et L. Brisoux
CRIL – Université d'Artois
rue de l'Université – S.P. 16
62307 Lens Cedex – FRANCE
{sais, brisoux}@cril.univ-artois.fr

Résumé

Les nombreux travaux autour du problème SAT qui ont été effectués ces dernières années ont permis de se faire une idée assez précise des classes d'algorithmes à mettre en œuvre pour résoudre des requêtes exprimées en calcul propositionnel. La plupart de ces algorithmes demandent un codage de l'ensemble de clauses étudié par une matrice creuse. Dans cet article, nous étudions dans quelle mesure le formalisme d'expression des problèmes peut être étendu sans changer significativement ni les algorithmes de résolution, ni surtout les structures de données. L'extension que nous proposons est double : des formules à deux niveaux aux circuits à un nombre arbitraire de niveaux, des clauses aux fonctions de seuils. Nous montrons que tous les algorithmes classiques (filtrages, algorithmes énumératifs, méthodes stochastiques et diagramme binaire de décision) s'appliquent quasiment à l'identique sur cette extension.

1 Introduction

Depuis quelques années, des efforts importants ont été faits pour mettre au point un corpus d'algorithmes efficaces pour résoudre des requêtes posées sur des formules du calcul propositionnel. Bien entendu, le terme efficace doit être compris relativement à la difficulté intrinsèque des questions posées. Tester la satisfiabilité d'un ensemble de clauses est et reste un problème difficile. Cependant, on commence à avoir une idée assez précise du type de méthodes à mettre en œuvre pour réduire, autant que faire se peut, cette difficulté. On peut diviser ces algorithmes en quatre catégories (par ordre croissant de complexité[1]) :

–Les filtrages. Il s'agit principalement de la propagation des littéraux purs et unitaires en temps linéaire [42, 19, 50]. Ils sont destinés à résoudre des problèmes dans P : 2SAT, Horn-SAT, Horn-renomabilité, ... [20, 21, 28, 29].

–Les algorithmes de recherche stochastique, au premier rang desquels GSAT [54] et ses multiples variantes [52, 53]. Ils sont destinés à chercher une solution au problème de satisfiabilité. Ils traitent donc un problème dans NP.

–La procédure de Davis et Putnam [22] et ses variantes [5, 23, 34, 40] qui trouvent une solution au problème de satisfiabilité quand il y en a une et montrent qu'il n'y en a pas dans le cas contraire. Ces algorithmes résolvent donc un problème dans coNP[2].

–Les diagrammes binaires de décision [9, 8]. Ils sont utilisés pour le calcul d'impliquants premiers [17, 48], la minimisation à deux niveaux [15], le calcul des états accessibles d'un automate d'états fini [16, 10], l'évaluation de la fiabilité [48]. Ces problèmes sont en général dans PSPACE ou #P.

Les améliorations apportées à/par ces algorithmes sont dues principalement à la mise au point de structures de données efficaces. C'est clair pour les filtrages et pour les diagrammes binaires de décision. Qui a programmé une procédure de Davis et Putnam ou une méthode de réparation locale sait que l'efficacité pratique de ce type d'algorithmes est pour beaucoup une question de codage intelligent des clauses. Il n'est pas dans notre propos de minimiser l'importance des heuristiques — l'autre grand sujet de discussion [34] — mais le lecteur conviendra que la pertinence de celles-ci est très dépendante du type de formules étudiées[3]. De plus, si l'on s'intéresse non plus à des heuristiques individuelles mais à des classes d'heuristiques on retombe, mutatis mutandis, sur une discussion sur les structures de données permettant de les mettre en œuvre.

À l'exception des diagrammes binaires de décision, les algorithmes cités plus haut travaillent tous sur des ensembles de clauses (formules CNF ou DNF). Quelques articles ont proposé des extensions vers des formules non clausales (par exemple [57, 19, 14]), mais cette voie n'a pas été systématiquement explorée. Or tous les problèmes que l'on peut modéliser en calcul propositionnel ne se codent pas naturellement par des formules CNF ; loin s'en faut. Il existe bien sûr des algorithmes efficaces permettant de passer d'une formule générale à une formule CNF équivalente pour la satisfiabilité [47, 59], mais d'une part cela se fait au détriment de la structure de la formule (qui peut contenir des informations intéressantes que les résolveurs peuvent exploiter, e.g. des symétries), d'autre part cette équivalence ne tient que pour la satisfiabilité.

De plus, certains problèmes se codent mal avec des ensembles de clauses. Par exemple, la satisfiabilité des formules de Tseitin [56] peut être prouvée efficacement si ces dernières sont représentées par une conjonction de chaînes d'equivalences (Biconditional Normal Form) alors que leur résolution est exponentielle dans le cas d'un codage par une CNF [24]. Il faut aussi noter que lors de la modélisation d'un problème réel, de nombreuses informations ne sont que des fonctions booléennes exprimant des dépendances entre variables. Cette notion de « dépendance » [38] explicite dans le cas du formalisme que nous proposons, est difficile à obtenir dans le cas CNF[35]. La reconnaissance et l'utilisation des dépendances entre variables

[1] Le lecteur pourra se reporter à [27] ou [46] pour une définition exacte des classes de complexité évoquées ci-après.

[2] Ils sont peuvent aussi être utilisés pour produire des impliquants et des impliqués premiers [44, 12, 51].

[3] C'est ce que montrent en particulier les travaux sur les variations des modèles de générations aléatoires [49, 13].

peut amener une réduction sensible du temps de recherche. En effet, l'affectation d'une variable indépendante détermine les valeurs des variables dépendantes, ce qui réduit le nombre d'interprétations possibles à 2^n où n est le nombre de variables indépendantes.

Dans cet article, nous étudions comment étendre au maximum le formalisme utilisé tout en :

- Restant dans le cadre du calcul propositionnel qui nous semble adapté à la résolution de nombreux problèmes pratiques.
- Gardant les mêmes structures de données, ou en tout cas les mêmes principes de mise en œuvre des algorithmes cités plus haut.
- Gardant la même complexité algorithmique sur les formules étendues que sur les formules CNF et surtout la même efficacité pratique.

Plus précisément, nous montrons que cela est possible sur des circuits combinatoires à plusieurs niveaux et construits avec des fonctions de seuils.

Le reste de cet article est organisé comme suit : la prochaine section décrit formellement les circuits combinatoires considérés. Dans les sections suivantes, nous étudions l'extension des algorithmes de chacune des quatre catégories au nouveau formalisme (respectivement dans les sections 3 pour les filtrages, 4 pour les algorithmes énumératifs, 5 pour les méthodes de recherche locale et 6 pour les diagrammes binaires de décision). Finalement, nous discutons les perspectives dans la section 7.

2 L'extension proposée

Le formalisme que nous étudions dans cet article étend les ensembles de clauses dans deux directions :
–Des formules à deux niveaux (conjonction de disjonctions) aux formules à un nombre arbitraire de niveaux.
–Des clauses aux fonctions de seuils.

Syntaxe Le langage propositionnel $\mathcal{L}$ que nous considérons est construit sur un ensemble infini dénombrable de variables propositionnelles $\mathcal{V}$, l'ensemble des entiers naturels $\mathbb{N}$, et les symboles auxiliaires "=", "#", "(", ")", "∈", "," "[" et "]". $\mathcal{L}$ utilise deux constructions syntaxiques : les portes et les circuits.

L'ensemble des portes de $\mathcal{L}$ est le plus petit ensemble tel que : si $s, e_1, \ldots, e_k$ sont des variables de $\mathcal{V}$, $\alpha_1, \ldots, \alpha_k$, β_{min} et β_{max} sont des entiers positifs ou nuls, alors

$$s \;=\; \#(\alpha_1.e_1, \ldots, \alpha_k.e_k) \in [\beta_{min}, \beta_{max}] \qquad (1)$$

est une porte. On note $Var(P)$, l'ensemble des variables apparaissant dans la porte P. s est la sortie (le fanout) de la porte, les e_i sont ses entrées (fanins). On étend ces notations aux ensembles de portes : soit C un ensemble de portes, l'ensemble des variables apparaissant dans C est noté $Var(C)$, le sous-ensemble des variables n'apparaissant pas en membre droit d'une porte est noté $Sorties(C)$ et le sous-ensemble des variables n'apparaissant pas en membre gauche d'une porte est noté $Entrées(C)$. Notons que, dans un ensemble de portes certaines variables ne sont ni des entrées, ni des sorties.

L'ensemble des circuits de $\mathcal{L}$ est le plus petit ensemble d'ensembles de portes tel que :
–L'ensemble vide est un circuit (noté $\top$).
–Si C_1 et C_2 sont des circuits et $Var(C_1) \cap Var(C_2) = \emptyset$, alors $C_1 \cup C_2$ est un circuit.
–Si C est un circuit et P est une porte dont la sortie s n'appartient pas à $Var(C)$, alors $C \cup \{P\}$ est un circuit.

On remarque que cette définition assure que, dans un circuit, toute variable est la sortie d'au plus une porte. Un circuit C peut être vu comme un (hyper)graphe orienté acyclique dont les sommets sont les variables et dont les (hyper)arcs relient les entrées de chaque porte avec sa sortie. Dans le cadre de cet article, nous nous restreindrons au cas où ce graphe n'a qu'une racine, c'est-à-dire où le circuit n'a qu'une sortie. De plus, nous dirons indifféremment équation ou porte et formule ou circuit. Les seules "vraies" variables d'un circuit sont ses entrées. Les autres sont des variables intermédiaires qui peuvent être vues comme des noms pour les sous-formules (ou la formule elle-même).

Il est important d'étendre la définition des circuits donnée ci-dessus de façon à permettre la substitution de valeurs aux variables. Soient C un circuit, $x_1, \ldots, x_k$ des variables et $v_1, \ldots, v_k$ des valeurs booléennes (0 ou 1). On note $C_{x_1 \leftarrow v_1, \ldots, x_k \leftarrow v_k}$ le circuit C dans lequel les valeurs $v_1, \ldots, v_k$ sont substituées respectivement aux occurrences des variables $x_1, \ldots, x_k$.

Sémantique Une affectation σ est une application de $\mathcal{V}$ dans $\{0,1\}$. On étend par induction structurelle, les affectations en des applications de $\mathcal{L}$ dans $\{0,1\}$ de la façon suivante :

Soient σ une affectation et P une porte de la forme (1) alors $\sigma(P) = 1$ si et seulement si l'égalité $\sigma(s) = 1$ et la double inégalité $\beta_{min} \leq \sum_{i=1,..,k} \alpha_i.\sigma(e_i) \leq \beta_{max}$ sont vérifiées ou infirmées simultanément (si x est une constante $\sigma(x) = x$).

Soient σ une affectation et C un circuit alors $\sigma(C) = 1$ si et seulement si σ donne la valeur 1 à toutes les portes de C.

Un circuit est satisfiable s'il existe une affectation le satisfaisant, c'est une tautologie si toute affectation le satisfait et c'est une antilogie si aucune affectation le satisfait.

Autres notions classiques Les notions de littéral, de produit (ensemble fini de littéraux interprétés conjonctivement), de clauses (ensemble fini de littéraux interprétés disjonctivement), d'impliquants premiers et d'impliqués premiers ont leur signification usuelle. Il y a une correspondance bi-univoque entre les substitutions et les produits : à la substitution $x_1 \leftarrow 1, \ldots, x_m \leftarrow 1, y_1 \leftarrow 0, \ldots, y_n \leftarrow 0$ correspond le produit $\{x_1, \ldots, x_m, \neg y_1, \ldots, \neg y_n\}$ et réciproquement. Les produits et les substitutions peuvent aussi être vus comme des affectations partielles.

Puissance d'expression Il est clair que $\mathcal{L}$ est une extension des formules CNF. En particulier, si a et b sont des variables, les formules $(a \vee b)$, $(a \wedge b)$ et $\neg a$ seront codées par les équations suivantes :

$$a.ou.b \;=\; \#(a,b) \in [1,2] \qquad (2)$$
$$a.et.b \;=\; \#(a,b) \in [2,2] \qquad (3)$$
$$non.a \;=\; \#(a) \in [0,0] \qquad (4)$$

De la même façon, $\mathcal{L}$ est une extension des systèmes d'inéquations linéaires en variables 0/1. Soit l'inéquation linéaire E suivante :

$$\alpha_1.x_1 + \ldots + \alpha_m.x_m - \beta_1.y_1 - \ldots - \beta_n.y_n \leq \gamma$$

Pour la coder dans $\mathcal{L}$, on commence par coder les littéraux négatifs :

$$\Sigma_{i=1,m}\alpha_i.x_i + \Sigma_{j=1,n}\beta_j.(1-y_j) \leq \gamma + \Sigma_{i=1,n}\beta_i$$

Puis, on écrit directement les portes correspondantes :

$$\begin{aligned} E &= \#(\alpha_1.x_1,\ldots,\alpha_m.x_m,\beta_1.z_1,\ldots, \\ &\qquad \ldots\beta_n.z_n) \in [0,\gamma+\Sigma_{i=1,n}\beta_i] \\ z_1 &= \#(y_1) \in [0,0] \\ &\vdots \\ z_n &= \#(y_n) \in [0,0] \end{aligned}$$

Notons qu'en pratique la négation peut être codée directement (sans introduire de nouvelles variables), en utilisant des littéraux plutôt que des variables lors du codage d'une porte (1).

Les liens entre la programmation linéaire en $\{0,1\}$ et le calcul propositionnel ont déjà été étudiés, en particulier par Hooker (voir par exemple [32, 31, 33]). Cependant, Hooker a surtout examiné comment interpréter les algorithmes de programmation linéaire dans le cadre de la logique booléenne. Nous proposons au contraire d'adapter les méthodes développées pour le calcul propositionnel à la programmation linéaire en $\{0,1\}$ (de plus nous considérons des formules à un nombre arbitraire de niveaux, et non des systèmes d'inéquations).

Signalons tout de suite quelques avantages de l'extension proposée sur les formules CNF.

–Elle est beaucoup plus concise. Par exemple, le fait que k formules parmi n doivent être vraies se code en une seule équation.

–Elle permet facilement l'introduction de lemmes, c'est à dire de variables codant des formules. Les lemmes permettent d'abord de construire des preuves courtes pour des formules intraitables autrement [56, 11, 4], mais surtout de compiler facilement des descriptions de plus haut niveau (par exemple des diagrammes de fiabilité [45]).

–Elle permet de tirer parti facilement des symétries ou de toutes autres régularités apparaissant dans les problèmes, en les détectant automatiquement comme dans [5] ou en écrivant directement les formules de façon à en éliminer les redondances[4].

–Elle permet de traiter au même plan une formule et sa négation, ce qui est intéressant dans le cadre de la compilation compositionnelle de formalismes de haut niveau.

–Elle permet de mettre en œuvre des procédures de réécriture des formules dans l'esprit du vérificateur de modèle INSTEP [58]. INSTEP implémente deux types de règles : les filtrages (ou simplifications immédiates) des formules — nous reviendrons sur ce sujet dans la prochaine section — et la décomposition de Shannon qui consiste à remplacer une formule f par la formule $(x \wedge f_{x\leftarrow 1}) \vee (\neg x \wedge f_{x\leftarrow 0})$. Ce type de réécriture peut être utilisé en tant que tel pour résoudre le problème posé, ou comme pré-traitement des formules. Cette dernière voie semble particulièrement prometteuse pour les algorithmes utilisant les diagrammes binaires de décision.

Il reste bien entendu à montrer que cette extension ne se fait pas au prix d'une complexité accrue des algorithmes. C'est ce que nous allons faire dans les prochaines sections.

[4]Par exemple en imposant, dans un problème de Ramsey comme ceux traités dans [44], des inéquations sur le nombre d'arêtes de chaque couleur.

3 Filtrages

La première chose à vérifier est que le formalisme proposé peut être codé dans des structures de données permettant de propager efficacement les affectations de valeurs aux variables et de détecter les variables astreintes à prendre une valeur unique. Ces deux opérations sont la clé de voûte de toute procédure énumérative efficace, et par là, de la mise au point d'algorithmes optimums sur les classes polynômiales principales du problème SAT et des problèmes de satisfaction de contraintes [28]. La première propage des valeurs des entrées vers la sortie du circuit, la seconde procède en sens contraire. La structure de données doit donc permettre ce double mouvement, de haut en bas et de bas en haut.

Opérateur de conséquence immédiate Nous allons dans un premier temps définir un opérateur de conséquence immédiate $\vdash$ (qui décrit formellement les opérations mentionnées ci-dessus), puis nous proposerons des structures de données pour le mettre en œuvre efficacement.

La figure 1 présente $\vdash$ à l'aide de séquents. La signification de chacune de ces règles est suffisamment explicite pour que nous ne la décrivions pas en détails ici. Les séquents de $\vdash$ transforment un couple (circuit, substitution) en un autre couple (circuit, substitution) équivalent :

$$C_\sigma \vdash C'_{\sigma'} \quad \Rightarrow \quad C_\sigma \equiv C'_{\sigma'} \tag{5}$$

Nous noterons $\vdash^*$ l'opération consistant à appliquer un nombre quelconque de fois l'une des règles décrites figure 1. Il est bien clair que l'opérateur $\vdash$ admet un point fixe unique (quel que soit l'ordre d'application des règles). Nous noterons $\downarrow^{\vdash} C_\sigma$ ce point fixe.

Structure de données La structure de données que nous proposons (pour mettre en œuvre $\vdash$) est la suivante. On construit :

- La liste des portes du circuit étudié.
- La liste des entrées de chaque porte.
- Pour chaque variable, la liste des portes dont elle est une entrée.

Les deux dernières listes sont en fait des listes de pointeurs vers des monômes $\alpha_i.x_i$ qui comportent chacun un pointeur vers la variable x_i et un pointeur vers la porte dans laquelle ils apparaissent. La structure codant une variable doit aussi contenir un pointeur vers la porte dont elle est éventuellement la sortie. Cette structure est une simple généralisation des matrices creuses utilisées pour coder les ensembles de clauses [50].

Afin de coder l'affectation partielle courante, on maintient la valeur de chaque variable (et de chaque porte). On maintient aussi, pour chaque porte de la forme (1), la valeur minimum γ_{min} et la valeur maximum γ_{max} que peut prendre la somme des $\alpha_i.x_i$ dans l'affectation partielle courante. Ces valeurs correspondent respectivement à l'affectation à 0 et à 1 des variables non encore affectées par σ. Quand une entrée reçoit une valeur, γ_{min} et γ_{max} sont mis à jour (en temps constant dans la mesure où il suffit d'ajouter ou de soustraire un α aux valeurs précédentes des γ). Si $\gamma_{min} > \beta_{max}$ ou si $\gamma_{max} < \beta_{min}$, la sortie s prend la valeur 0 et la propage. Si $\gamma_{min} \geq \beta_{min}$ et $\gamma_{max} \leq \beta_{max}$, la sortie s prend la valeur 1 et la propage.

$$\perp : \frac{C \qquad x \leftarrow 1, x \leftarrow 0, \sigma}{\perp}$$

$$\leftarrow_0 : \frac{C, s = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad x_i \leftarrow 0, \sigma}{s = \#(\alpha_1.x_1, \ldots, \alpha_{i-1}.x_{i-1}, \alpha_{i+1}.x_{i+1}, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}]}$$

$$\leftarrow_1 : \frac{C, s = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad x_i \leftarrow 1, \sigma}{s = \#(\alpha_1.x_1, \ldots, \alpha_{i-1}.x_{i-1}, \alpha_{i+1}.x_{i+1}, \ldots, \alpha_k.x_k) \in [\beta_{min} - \alpha_i, \beta_{max} - \alpha_i]}$$

$$\uparrow^1 : \frac{C, s = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \beta_{min} \leq 0, \;\; \Sigma_{i=1}^k \alpha_i \leq \beta_{max}}{s \leftarrow 1}$$

$$\uparrow^0_{min} : \frac{C, s = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \Sigma_{i=1}^k \alpha_i < \beta_{min}}{s \leftarrow 0}$$

$$\uparrow^0_{max} : \frac{C, s = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \beta_{max} < 0}{s \leftarrow 0}$$

$$\downarrow_1^{min} : \frac{C, 1 = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \Sigma_{j=1,..,k, i \neq j} \alpha_j < \beta_{min}}{x_i \leftarrow 1}$$

$$\downarrow_0^{max} : \frac{C, 1 = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \alpha_i > \beta_{max}}{x_i \leftarrow 0}$$

$$\downarrow_0^{min} : \frac{C, 0 = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \alpha_i \geq \beta_{min}, \;\; \Sigma_{j=1}^k \alpha_j \leq \beta_{max}}{x_i \leftarrow 0}$$

$$\downarrow_1^{max} : \frac{C, 0 = \#(\alpha_1.x_1, \ldots, \alpha_k.x_k) \in [\beta_{min}, \beta_{max}], \qquad \Sigma_{j=1,..,k, i \neq j} \alpha_j \leq \beta_{max}, \;\; \beta_{min} \leq 0}{x_i \leftarrow 1}$$

FIG. 1: Les règles définissant l'opérateur de conséquence immédiate $\vdash$.

Les affectations de valeurs aux variables se font donc sans modifier la structure de données. On utilise une pile pour mémoriser les affectations à effectuer.

Dans le cas où c'est la variable s qui est affectée, γ_{min} et γ_{max} sont utilisés pour effectuer les tests correspondants aux séquents $\downarrow_1^{min}$, $\downarrow_0^{min}$, $\downarrow_1^{max}$ et $\downarrow_0^{max}$ de la figure 1. On note que si la valeur de x_i est fixée par propagation alors toute variable x_j telle que $\alpha_j \geq \alpha_i$ est astreinte à prendre la même valeur. L'idée est donc de maintenir une liste des x_i triée par ordre décroissant des α_i ainsi qu'un pointeur vers la première variable non affectée de la liste. Ce tri initial de la liste se fait en $\mathcal{O}(k.log(k))$ (c'est un pré-traitement).

Le calcul du point fixe de l'opérateur de conséquence immédiate revient à étendre la substitution courante. Il se fait en temps linéaire amorti [55].

Lemme 1 (Complexité de la propagation) *Soient C un circuit et σ une substitution codés dans les structures de données décrites ci-dessus. Le calcul de $\downarrow^{\vdash} C_\sigma$ est en $\mathcal{O}(|C|)$, où $|C|$ dénote le nombre de monômes $\alpha_i.x_i$ présents dans le circuit C.*

Pour vérifier le lemme précédent, il suffit de remarquer que chaque monôme est parcouru au plus deux fois : une fois « verticalement » lors de l'affectation de la variable correspondante et une fois « horizontalement » si la variable est contrainte à prendre une valeur donnée.

Affectations libres Soient C un circuit, x une variable et v une valeur (0 ou 1). On dit que l'affectation de v à x est libre si pour toute affectation σ satisfaisant C et telle que $\sigma(x) = 1 - v$, l'affectation σ' égale partout à σ sauf en x satisfait C. Cette notion est sémantique. Il convient donc d'en donner une version affaiblie, syntaxique, qui pourra être détectée facilement sur la structure de données. La table 1 donne des conditions syntaxiques suffisantes (mais pas forcément nécessaires) pour qu'une affectation soit libre ou imposée. Le lecteur remarquera que la notion d'affectation imposée généralise celle de littéral unitaire et que la notion d'affectation libre généralise celle de littéral pur. Intuitivement, l'affectation de la valeur v à la variable x est imposée s'il existe une porte qui serait immédiatement falsifiée si la valeur $1-v$ était affectée à x. L'affectation de la valeur v à la variable x est libre si pour toutes les portes dans lesquelles x apparaît en entrée, cette affectation ne fait que renforcer la satisfaction potentielle de la porte. On voit donc qu'il y a une dualité profonde entre affectations libres et affectations imposées.

D'autre part, si l'affectation d'un des x_i à la valeur v_i est libre alors c'est le cas pour toutes les variables non affectées de la porte. Par conséquent, dès que la condition est remplie, on peut parcourir la liste et mettre à jour des compteurs qui pour chaque variable maintiennent le nombre de portes dans lesquelles elle n'est pas libre positivement et négativement. Dès qu'un de ces deux compteurs passe à 0, la variable peut être librement affectée à l'autre valeur. Ici encore, il n'est pas difficile de voir que la gestion des compteurs se fait en temps linéaire amorti sur la taille de l'instance.

Au coût du pré-traitement près (qui est très faible), on a donc des algorithmes de propagation des variables et de détection des sous-formules astreintes à prendre une valeur unique et des sous-formules dont la valeur est libre aussi efficaces que ceux dont on dispose grâce à la structure de matrice creuse pour les ensembles de clauses [50].

Extension de la notion d'« autark » La notion d'autark (ou de partition du modèle) [25, 43, 44] se généralise de différentes façons. La plus simple est sans doute la suivante : soient C un circuit et σ une affectation partielle, σ est un autark de C si aucune des variables affectées par

valeur de s	condition	affectation	caractère
$s=1$	$\gamma_{min}+\alpha_i > \beta_{max}$	$x_i \leftarrow 0$	imposée
$s=1$	$\gamma_{max}-\alpha_i < \beta_{min}$	$x_i \leftarrow 1$	imposée
$s=0$	$(\gamma_{min}+\alpha_i \geq \beta_{min}) \wedge (\gamma_{max} \leq \beta_{max})$	$x_i \leftarrow 0$	imposée
$s=0$	$(\gamma_{max}-\alpha_i \leq \beta_{max}) \wedge (\gamma_{min} \geq \beta_{min})$	$x_i \leftarrow 1$	imposée
$s=1$	$\gamma_{max} \leq \beta_{max}$	$x_i \leftarrow 1$	libre
$s=1$	$\gamma_{min} \geq \beta_{min}$	$x_i \leftarrow 0$	libre
$s=0$	$\gamma_{max} \leq \beta_{max}$	$x_i \leftarrow 0$	libre
$s=0$	$\gamma_{min} \geq \beta_{min}$	$x_i \leftarrow 1$	libre

TAB. 1: Table récapitulative des affectations imposées et libres

σ n'apparaît dans une équation non-satisfaite de C. En clair, σ sépare C en deux : des équations satisfaites et des équations dans lesquelles les variables affectées par σ n'apparaissent pas. Il est clair que si σ est un autark alors C est satisfiable si et seulement si C_σ l'est. La détection des autarks se fait suivant le même principe et avec le même coût (temps linéaire amorti) que sur les ensembles de clauses, c'est à dire en maintenant, pour chaque variable, un compteur du nombre d'équations non satisfaites dans lesquelle la variable apparaît [50].

Avec l'extension des notions de littéral unitaire, de littéral pur et d'autark, on dispose donc de toutes les briques permettant de mettre au point des algorithmes efficaces pour traiter les principales classes polynômiales du problème SAT. Cette question a été étudiée en détails dans [28, 29]. Notons que le fait de disposer d'un formalisme plus puissant devrait permettre d'étendre ces classes polynômiales.

Procédure *DPL(C)*
entrée : C un circuit (ensemble de portes)
sortie : vrai si C est satisfiable ; faux sinon
début
  R_1 *: règle de simplification*
  Filtrage et propagation
  si C est vide
    alors retourner vrai
  sinon si il existe une porte falsifiée
    alors retourner faux
  sinon début
    R_2 *: règle de séparation*
    choix de la variable x et de la valeur v à affecter
    si $DPL(C_{x \leftarrow v})$
      alors retourner vrai
    sinon retourner $DPL(C_{x \leftarrow 1-v})$
  fin
fin

FIG. 2: La procédure DPL

4 Algorithmes énumératifs

On peut distinguer deux grandes classes d'algorithmes complets décidant de la satisfiabilité d'une formule booléenne :

- Ceux appliquant sous une forme ou sous une autre le principe de résolution (c'est à dire travaillant par réécriture de la formule considérée).
- Ceux qui explorent (énumèrent) de façon systématique l'espace de recherche.

Cette dernière classe s'est avérée de loin la plus efficace en pratique. Des travaux récents ont montré que les mises en œuvre les plus efficaces ne sont que des variantes de la procédure bien connue de Davis, Logeman et Loveland (DPL)[22]. Notons, que cette procédure peut être adaptée pour compiler des bases de connaissances propositionnelles (e.g. en recherchant une couverture d'impliquants (impliqués) premiers) [51, 13, 7].

La procédure de DPL Dans le cadre de cet article, nous limitons notre étude à l'extension de la procédure DPL au formalisme proposé (voir Fig. 2).

La procédure DPL se décompose en deux règles essentielles : la règle de simplification (R_1) et la règle de séparation (R_2). Son extension aux circuits nécessite une adaptation de ces deux règles. L'extension de la règle de simplification a été discutée dans la section précédente.

Les heuristiques La sélection de la prochaine variable à affecter est un facteur important dans l'efficacité de la procédure énumérative DPL. Une bonne heuristique peut réduire sensiblement la taille de l'arbre de recherche. De nombreux travaux ont été consacrés à l'amélioration de la règle de séparation (choix de la variable de branchement). Citons parmi elles les plus récentes : C-SAT [23], Tableau [18], POSIT [26], Satz [41] et Relsat[2].

Dans le cadre CNF, l'heuristique la plus utilisée est sans doute celle proposée par Jeroslow et Wang [36] qui consiste à choisir la variable apparaissant le plus souvent dans les clauses les plus courtes. Plus précisément, la variable x_i choisie est celle maximisant $w(x_i)+w(\neg x_i)$, avec $w(x_i) = \sum_{C \in S, x_i \in C} w(C)$, et $w(C) = 2^{-|c|}$. Les nombreuses variantes proposées en suite diffèrent essentiellement par leur fonction poids.

Pour adapter cette heuristique, on définit dans un premier temps la fonction $w(P)$ associée à une porte P. Il est évident que la porte définie par l'équation (1) peut s'écrire sous forme d'une conjonction de deux inéquations linéaires ILP(6) et ILN(7) :

$$\sum_{1 \leq i \leq n} \alpha_i x_i \geq \beta_{min} \tag{6}$$

et

$$\sum_{1 \leq i \leq n} \alpha_i \neg x_i \geq (\gamma - \beta_{max}) \tag{7}$$

avec

$$\sum_{1 \leq i \leq n} \alpha_i = \gamma$$

On remarque alors que l'équation précédente fait apparaître à la fois les occurrences positives et négatives, on retrouve de manière évidente la notion classique de littéraux purs (i.e. x_i (respectivement $\neg x_i$) est pur si et seulement si $\forall P \in C$ tel que $x_i \in P$, $\beta_{max} = \gamma$ (respectivement $\beta_{min} = 0$)).

Ci-dessous, la description de la fonction $f(v)$:

Soit C un circuit,

$$f_w(v) = \sum_{\forall P \in C,\ v \in P} w(P) \tag{8}$$

où $w(P)$ est défini de la façon suivante[5] :

- si $s = 1$, $w(P) = \begin{cases} w(ILP) & \text{si } v = x \\ w(ILN) & \text{si } v = \neg x \end{cases}$
- si $s = 0$, $w(P) = \begin{cases} w(ILN) & \text{si } v = x \\ w(ILP) & \text{si } v = \neg x \end{cases}$

Il reste à définir le poids associé à une inéquation linéaire de la forme :

$$\sum_{1 \leq i \leq n} \alpha_i x_i \geq \beta \tag{9}$$

Il est à noter que la transformation (sans introduction de nouvelles variables) d'une inéquation linéaire en CNF peut amener à un nombre exponentiel de clauses [30]. Pour se rapprocher le plus possible des poids utilisés dans le cadre CNF pour pondérer une clause, nous avons juste besoin de connaître le nombre et la taille des clauses obtenues. Pour ce faire nous utilisons une propriété définie dans [3], montrant la transformation d'une formule de cardinalité en clauses. Nous rappelons qu'une formule de cardinalité peut être considérée comme une inéquation où tous les coefficients ont une valeur égale à un. Dans notre cas, nous considérons le coefficient associé à une variable comme une indication du nombre de répétitions de la variable. Nous obtenons donc pour l'inéquation linéaire (9) une approximation du nombre et de la longueur des clauses :

- la longueur des clauses est : $\gamma - \beta + 1$,
- le nombre de clauses est : $C_\gamma^{\gamma-\beta+1}$

avec $\gamma = \sum_{1 \leq i \leq n} \alpha_i$.

Nous proposons ci-dessous, les différentes fonctions de pondération d'une inéquation linéaire :

$$w_1 = (\alpha_i \times C_\gamma^{\gamma-\beta+1}) \times 2^{-(\gamma-\beta+1)}$$

$$w_2 = (\frac{\alpha_i}{\gamma - \beta + 1}) \times 2^{-(\gamma-\beta+1)}$$

Rappelons que dans le cas CNF, une clause C est pondérée par un poids de $2^{|C|}$. Dans les deux pondérations précédentes, $|C|$ est remplacé par $\gamma - \beta + 1$ qui est une généralisation évidente. Le facteur multiplicatif permet d'accentuer notre préférence pour les inéquations linéaires nécessitant de nombreux coefficients pour les satisfaire.

La fonction de calcul du score d'une variable x est obtenue par :

$$F_{w_i}(x) = f_{w_i}(x) + f_{w_i}(\neg x) + \alpha \times \min(f_{w_i}(x), f_{w_i}(\neg x)) \tag{10}$$

(avec $\alpha = 1.5$ [6])

[5] Les variables du circuit apparaissant uniquement à gauche du symbole d'égalité sont affectées à 1.

5 Recherche stochastique

Les techniques de recherche stochastique consistent à tester différentes affectations des variables. Les affectations à tester sont obtenues en modifiant de proche en proche une affectation initiale (en général en inversant la valeur d'une variable, d'où le nom de réparation locale donné aussi à ces méthodes). L'affectation initiale ainsi que les modifications sont obtenues plus ou moins pseudo-aléatoirement : on cherche à faire les meilleurs choix possibles tout en laissant un degré de liberté afin d'assurer une bonne couverture des affectations possibles. Ces méthodes incomplètes ont été utilisées avec succès pour résoudre le problème SAT [54]. Le schéma général d'une procédure de recherche stochastique est décrit Fig. 3.

Procédure *RL(C)*
entrée : C un circuit (ensemble de portes)
sortie : Un modèle, si trouvé
  pour i **allant de** 1 **à** max. essais **faire**
    choisir une affectation initiale I
    pour j **allant de** 1 **à** max. réparations **faire**
      si I satisfait C
        alors retourner vrai
      sinon
        choix de la variable x à inverser
        $I = I$ avec x inversée
      finsi
    finpour
  finpour
  retourner faux
fin

FIG. 3: Procédure générique de recherche stochastique

L'adaptation de la procédure précédente aux circuits ne présente pas de difficulté majeure. On remarque tout d'abord que les variables peuvent prendre deux valeurs (vrai ou faux), alors qu'elles peuvent en prendre trois dans les procédures énumératives (vrai, faux ou non affectée). Quoi qu'il en soit, la modification de la valeur d'une variable se fait de la même façon dans les deux types de procédure. La partie la plus délicate de l'algorithme concerne la sélection des variables candidates à l'inversion, parmi lesquelles une est choisie pseudo-aléatoirement. Cela demande d'attribuer une note à chaque variable, et de maintenir les notes le plus efficacement possible. Une extension naturelle du calcul des notes mis en œuvre dans GSAT [54] pourrait être la suivante : Soient I l'interprétation courante et x une variable de C, alors $w(x) = w_+(x) - w_-(x)$, où $w_+(x)$ (resp. $w_-(x)$) dénote le nombre de portes de C actuellement falsifiées (resp. satisfaites) par I et qui deviendraient satisfaites (resp. falsifiées), si la valeur de x était inversée. Les compteurs $w_+(x)$ et $w_-(x)$ peuvent être mis à jour efficacement en utilisant la même technique que pour la détection des littéraux unitaires (voir section 3).

6 Diagrammes Binaires de Décision

Le calcul du BDD codant une équation de la forme (1) ne pose pas de problème particulier. On calcule les BDDs

E_i codant les e_i, puis on applique un algorithme énumératif (de type "forward checking") qui sépare sur la valeur des e_i et construit les BDDs en remontant. Par exemple, si l'on sépare sur la valeur de e_1 (codé par le BDD E_1) et que les BDDs construits pour $e_1 = 1$ et $e_1 = 0$ sont respectivement R_1 et R_0, le BDD codant la formule sera $(E_1 \wedge R_1) \vee (\neg E_1 \wedge R_0)$. Or, la procédure de calcul du BDD codant la fonction $(f \wedge g) \vee (\neg f \wedge h)$ à partir des BDDs codant respectivement f, g et h est précisément la procédure centrale de paquetages BDD [8].

En pratique, on a tout intérêt à mémoriser les résultats intermédiaires de façon à ne pas refaire deux fois le même calcul. Supposons par exemple que l'on veuille calculer le BDD associé à l'équation :

$$\begin{aligned} s &= \#(\alpha_1.e_1, \alpha_2.e_2, \alpha_3.e_3, \alpha_4.e_4, && (11) \\ & \qquad \ldots, \alpha_k.f_k) \in [\beta_{min}, \beta_{max}] && (12) \end{aligned}$$

Supposons de plus que $\alpha_1 + \alpha_2 = \alpha_3 + \alpha_4$. Alors le même résultat intermédiaire sera obtenu pour les affectations $e_1 \leftarrow 1$, $e_2 \leftarrow 1$, $e_3 \leftarrow 0$, $e_4 \leftarrow 0$, et $e_1 \leftarrow 0$, $e_2 \leftarrow 0$, $e_3 \leftarrow 1$, $e_4 \leftarrow 1$. Il est donc inutile de calculer deux fois le BDD correspondant. On a là un cas typique d'utilisation des chaînes d'addition de Knuth [39].

7 Conclusion

Le calcul propositionnel est un excellent compromis entre puissance d'expression et efficacité des algorithmes de résolution associés. Cependant, la plupart des algorithmes proposés dans la littérature travaillent sur des ensembles de clauses (formule CNF). Or, tous les problèmes en calcul propositionnel ne se codent pas naturellement par des formules CNF, et le passage vers une forme CNF fait perdre des informations structurelles importantes.

Dans cet article, nous avons montré qu'en considérant des circuits combinatoires à plusieurs niveaux et construits avec des fonctions de seuil, il est possible d'étendre au maximum le formalisme sans changer (ou presque) les algorithmes, et surtout les structures de données. En considérant les classes d'algorithmes, nous avons ainsi montré que leur adaptation au formalisme proposé ne présente pas de difficulté particulière.

Ce travail est un premier pas vers la mise en place d'un formalisme booléen plus général. Il ouvre de nombreuses perspectives (dont certaines sont en cours d'étude) :

- la validation de l'approche sur des problèmes réels,
- la mise en place d'heuristiques pour prendre en compte certaines informations structurelles (e.g. les dépendances,...),
- la mise en place de techniques de compilation de formules booléennes (recherche de couverture d'impliquants (impliqués) premiers).
- l'extension des algorithmes pour résoudre des problèmes d'optimisation (ce que permettent les poids associés aux variables dans les équations).

⋮

–

Références

[1] P. Barth. A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimisation. Technical Report MPI-I-95-2-003, Max-Planck-Institute für informatik, January 1995.

[2] R.J. Bayardo Jr. and R.C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proc. of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208, Providence, Rhode Island, 27-31 July 1997.

[3] B. Benhamou and L. Saïs. Two proof procedures for a cardinality based language in propositional calculus. In P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *STACS*, pages 71–82, Caen, France, February 1994. Springer-Verlag.

[4] W. Bibel. Short Proofs of the Pigeonhole Formulas Based on the Connection Method. *Journal of Automated Reasoning*, 6, 1990.

[5] A. Billionnet and A. Sutter. An efficient algorithm for the 3-satisfiability problem. *Operation Researchs Letters*, 12(1) :29–36, 1992.

[6] Y. Boufkhad. *Aspects probabilistes et algorithmiques du problème de satisfiabilité*. PhD thesis, Université de Paris VI, 1996.

[7] Y. Boufkhad, É. Grégoire, P. Marquis, B. Mazure, and L. Saïs. Tractable Cover Compilations. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 122–127, Nagoya, Japan, August 1997.

[8] K. Brace, R. Rudell, and R. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*. IEEE 0738, 1990.

[9] R. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8) :677–691, August 1986.

[10] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *Proceedings of the 27th ACM/IEEE Design Automation Conference, DAC'90*, pages 16–51, June 1990.

[11] S.R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52 :916–927, 1987.

[12] T. Castell and M. Cayrol. Arguments for the Easy Generation of Random, Hard and Satisfiable Instances of SAT (Easy to be Hard II). In *Proceedings of the ECAI'96 post-conference Workshop on SAT*, 1996.

[13] T. Castell and M. Cayrol. Une nouvelle méthode de calcul des implicants et des impliqués premiers. In *Actes des Journées sur la Résolution Pratique de Problèmes NP-Complets, CNPC'96*, pages 153–169, Teknea, 1996.

[14] P. Codognet and D. Diaz. A Simple and Efficient Boolean Solver for Constraint Logic Programming. *Journal of Automated Reasoning*, 17 :97–128, 1996.

[15] O. Coudert. Two-level minimization : an overview. *INTEGRATION, the VLSI Journal*, 17 :97–140, October 1994.

[16] O. Coudert, C. Berthet, and J-C. Madre. Verification of Synchronous Sequential Machines Based on Symbolic Execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407, pages 365–373. LNCS, 1989.

[17] O. Coudert and J.-C. Madre. Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In *Proceedings of the 29th ACM/IEEE Design Automation Conference, DAC'92*, June 1992.

[18] J. M. Crawford and L. D. Auton. Experimental Results on the Crossover Point in Random 3-SAT. *Artificial Intelligence*, 81 :31–57, 1996.

[19] M. Dalal. Efficient Propositional Constraint Propagation. In *Proc. of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 409–414, San Jose, California, 12-16 July 1992.

[20] M. Dalal. Tractable Deduction in Knowledge Representation Systems. In *Proceedings of the 3^{th} International Conference on Principles of Knowledge Representation and Reasoning, KR'92*, pages 393–402, 1992.

[21] M. Dalal and D. W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44(4) :173–180, 10 December 1992.

[22] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *CACM*, 5 :394–397, 1962.

[23] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In *Second DIMACS Implementation Challenge*, 1994.

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

[25] S. Even, A. Itai, and A. Shamir. On the Complexity of Timetable and Multicommodity Flow Problems. *SIAM J. Comput.*, 5 :691–703, 1976.

[26] J.W. Freeman. *Improvement to propositional satisfiability search.* PhD thesis, Univ. of Philadelphia, 1995.

[27] M.R. Garey and D.S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, San Fransisco, 1979.

[28] R. Génisson and A. Rauzy. Aspects algorithmiques des classes polynomiales du problème sat et des problèmes de satisfaction de constraintes. In *Actes du $10^{ième}$ congrès Reconnaissance de Formes et Intelligence Artificielle, RFIA'96*, pages 97–108, AFECT-AFIA, 1996.

[29] R. Génisson and A. Rauzy. Efficient Horn renaming : yet a Davis and Putnam's procedure. In *Résolution Pratique de Problèmes NP-Complets, CNPC'96*, pages 169–184, Teknea, 1996. Also, Technical Report 1092-95, LaBRI – URA CNRS 1304 – Université Bordeaux-I, 1995.

[30] F. Granot and P.L. Hammer. On the use of boolean functions in 0-1 programming. In *Methods of Operations Research*, volume 12, pages 154–184, 1971.

[31] J.N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4 :45–69, 1988.

[32] J.N. Hooker. Resolution vs. cutting plane solution of inference problems in propositional logic : some computational experience. *Operation Research Letter*, 7(1) :1–7, 1988.

[33] J.N Hooker. Generalized resolution for 0-1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6 :271–286, 1992.

[34] J.N. Hooker and V. Vinay. Branching Rules for Satisfiability. *Journal of Automated Reasoning*, 15 :359–383, 1995.

[35] T. Ibaraki, A. Kogan, and K. Makino. Functional Dependencies in Horn Theory. Technical Report 98-18, DIMACS, March 1998.

[36] R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. In *Annals of Mathematics and Artificial Intelligence*, pages 167–187, 1990.

[37] D.S. Johnson and M.A. Trick, editors. *Cliques, Coloring, and Satisfiability : Second Dimacs Implementation Challenge*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[38] H. Kautz, D. McAllester, and B. Selman. Exploiting Variable Dependency in Local Search. In *Abstracts of the Poster Sessions of IJCAI-97*, Nagoya, Japan, 1997.

[39] D.E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison Wesley, 1969.

[40] C. Li. Exploiting Yet More the Power of Unit Clause Propagation to Solve the 3-SAT Problem. In *Proceedings ECAI'96 Post-Conference Workshop on SAT*, 1996.

[41] C. Li and Anbulagan. Heuristic Based on Unit Propagation for Satisfiability Problems. In *Proc. of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya, Japan, 1997.

[42] M. Minoux. LTUR : A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and its Computer Implementation. *Information processing Letter*, 29 :1–12, 1988.

[43] B. Monien and E. Speckenmeyer. Solving Satisfiability in Less than 2^n Steps. *Discrete Applied Math*, 10 :287–295, 1985.

[44] L. Oxusoff and A. Rauzy. *L'Évaluation Sémantique en Calcul Propositionnel.* PhD thesis, GIA – Université de Aix-Marseille II, January 1989.

[45] A. Pagès and M. Gondrand. Fiabilité des systèmes. In *Collection de la Direction des Études et Recherches d'Électricité de France*, volume 39. Eyrolles, 1980. ISSN 0399-4198.

[46] C.H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1984.

[47] D.A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2 :293–304, 1986.

[48] A. Rauzy. New Algorithms for Fault Trees Analysis. *Reliability Engineering & System Safety*, 5(59) :203–211, 1993.

[49] A. Rauzy. On the Random Generation of 3-SAT-Instances. Technical Report 1060–95, LaBRI – URA CNRS 1304 – Université Bordeaux I, 1995.

[50] A. Rauzy. Polynomial restrictions of SAT : What can be done with an efficient implementation of the Davis and Putnam's procedure. In U. Montanari and F. Rossi, editors, *Proceedings of the International Conference on Principle of Constraint Programming, CP'95*, volume 976 of LNCS, pages 515–532, Springer Verlag, 1995.

[51] R. Schrag. Compilation for Critically Constrained Knowledge Bases. In *Proc. of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, volume 1, pages 510–515, Portland, Oregon, 4-8August 1996.

[52] B. Selman and H.A. Kautz. Domain Independent Extensions to GSAT : Solving Large Structured Satisfiability Problems. In *Proc. of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI'93)*, Chamberry, France, 1993.

[53] B. Selman, H.A. Kautz, and B. Cohen. Noise Strategie for Improving Local Search. In *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, volume 1, Seattle, Washington, 1-4August 1994.

[54] B. Selman, D. Mitchell, and H. Levesque. A New Method for Solving Hard Satisfiability Problems. In *Proc. of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, San Jose, California, 12-16 July 1992.

[55] R.E. Tarjan. Amortized Computational Complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2) :306–318, 1985.

[56] G.S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Structures in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1970.

[57] A. van Gelder. A Satisfiability Tester for Non-clausal Propositional Calculus. *Information and Computation*, 79 :1–21, 1988.

[58] F. Vlach. Simplification in a Satisfiability Checker for VLSI Applications. *Journal of Automated Reasoning*, 10 :115–136, 1993.

[59] J.M. Wilson. Compact normal forms in propositional logics and integer programming formulations. *Computer and Operatuons Research*, 90 :309–314, 1990.