Résolution sur des ensembles de clauses compressés : comment doper DP

Philippe Chatalic et **Laurent Simon**
Laboratoire de Recherche en Informatique,
U.M.R. CNRS 8623, Université Paris-Sud
91405 Orsay Cedex, France
email: `{chatalic,simon}@lri.fr`

Résumé

Cet article présente des opérateurs spécialisés dans la manipulation d'ensembles de clauses représentés à l'aide de ZBDD. Le pouvoir de compression de ces structures de données permet un codage efficace de certaines instances structurées et autorise le calcul d'opérations complexes, jusque-là impossibles, sur de très grands ensembles de clauses. Le principal opérateur, qui permet l'élimination d'une variable propositionnelle par coupure, est utilisé au sein du système ZRES, une implémentation de la procédure de Davis-Putnam dans sa version de 1960. ZRES peut représenter des ensembles de clauses de taille exponentielle par une structure de donnée polynômiale. Il est en mesure de résoudre deux problèmes considérés comme difficiles pour la résolution, Pigeon Hole et Urquhart, qui, tout deux, restent hors de portée des meilleurs démonstrateurs SAT actuels.

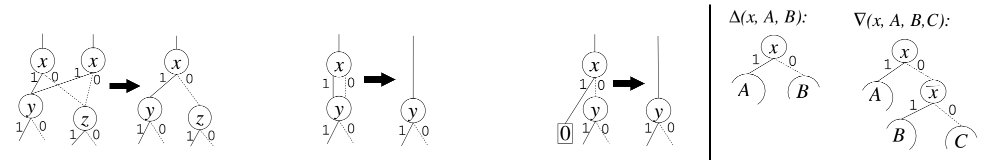
1 Introduction

Ces dernières années, nombre de travaux ont utilisé la logique propositionnelle comme cadre de travail, tant pour la résolution de problèmes que pour la représentation des connaissances. Son faible pouvoir d'expression, compensé en pratique par sa simplicité, permet de concevoir des démonstrateurs simples mais efficaces. Le problème SAT a ainsi été l'objet d'un renouveau d'intérêt, donnant lieu à des procédures complètes hautement optimisées [Zha97], aussi bien qu'à l'introduction de nouvelles et puissantes méthodes incomplètes [SLM92]. Dans le même temps, des instances aléatoires difficiles ont été identifiées [MSL92], permettant la découverte d'un phénomène de seuil, et des bibliothèques de benchmarks [Dim] ont été constituées, ce qui a entraîné de fructueuses compétitions de programmes. Si le rôle central de SAT dans la théorie de la complexité est probablement pour beaucoup dans la motivation de ces travaux, la résolution de problèmes concrets y tient également une part importante. La plupart des méthodes complètes actuelles pour résoudre SAT sont basés sur la procédure de Davis, Logeman et Loveland (DLL) [MDL62], dont le succès est dû en partie à sa faible consommation mémoire. Cette procédure ne faisant que des choix, elle n'effectue que des simplifications et des propagations unitaires. Le nombre de ces choix reste malgré tout exponentiel dans le pire des cas et seule l'utilisation d'heuristiques peut permettre de réduire la taille de l'espace de recherche. Le choix d'heuristiques efficaces est cependant rendu difficile dans la mesure où des procédures de choix judicieuses sur les exemples aléatoires ne le sont pas toujours pour des exemples structurés. Sur ces derniers, des heuristiques basées sur des techniques d'apprentissage semblent donner de meilleurs résultats [Zha97].

D'autres problèmes, comme le calcul d'impliquants/impliqués premiers ou la compilation de base de connaissances, nécessitent également de manipuler de grands ensembles de clauses. La difficulté vient alors du fait que la taille de ces ensembles peut croître de façon exponentielle. Dans tous les cas, on se trouve donc confronté à une explosion combinatoire. Un moyen de la contrecarrer peut être de prendre en compte la structure du problème d'une manière plus globale. Si on voulait faire un parallèle entre ces deux approches, pourtant bien différentes, on dirait que, si l'utilisation d'heuristiques permet de réduire efficacement la taille de l'espace de recherche, la taille des ensembles manipulés peut être réduite à l'aide d'algorithmes de *compression*. Dans les deux cas, la prise en compte globale du problème semble en effet payante.

Ce travail s'intéresse particulièrement aux structures de données utilisées pour représenter des ensembles de clauses dans le cas d'exemples structurés. Notre sentiment est que la structure inhérente de telles instances doit se traduire par des régularités dans sa représentation. Pour des formules sous forme normale conjonctive (*fnc*), de telles régularités peuvent correspondre à des sous-ensembles de littéraux communs à différentes clauses. Nous proposons d'utiliser une structure de données permettant la factorisation de séquences de littéraux communs, par rapport à un ordre donné. Les structures de *tries* [dK92] permettent de factoriser les clauses débutant par une même séquence de littéraux. Elles constituent encore aujourd'hui les structures les plus performantes utilisées dans les algorithmes de tests de subsomption [Zha97, dK92, CS00]. Nous généralisons cette approche, en permettant simultanément la factorisation des débuts et des fins de clauses communs. En pratique, nous utilisons une variante des BDD [Bry86], appelée ZBDD [Min93], afin de représenter un ensemble de clauses par sa fonction caractéristique. Nous montrons que certains ensembles de clauses de grande taille, correspondant à des instances structurées, peuvent ainsi être représentés de façon très compacte. De plus, cette structure est bien adaptée à la mise en œuvre des tests de subsomptions et des opérations ensemblistes classiques. Nous montrons que des opérations plus complexes peuvent être aussi réalisées et proposons un opérateur spécialisé dans le calcul de coupure d'une variable propositionnelle. Cet opérateur, qui travaille directement au niveau des ensembles de clauses, est utilisé au sein d'une implémentation de la procédure originelle de Davis et Putnam [DP60] (DP). Une analyse de ce système, appelé ZRES, est effectuée sur quelques exemples, considérés comme difficiles pour les procédures basées sur le principe de résolution.

La section suivante rappelle brièvement les principes des BDD et ZBDD, situe notre travail par rapport aux autres approches utilisant ces structures de données, et décrit ensuite les principes de l'opérateur de coupure. La troisième section montre comment cet opérateur peut être utilisé dans DP. La quatrième partie illustre le bien-fondé du terme de *compression* employé ici, puis étudie en détail le comportement de ZRES sur deux classes de problèmes difficiles pour la résolution, le problème des Pigeons et le problème d'Urquhart. Nous étudions comment ZRES résout ces instances, qui ne peuvent être résolues en pratique par les meilleurs démonstrateurs actuels de type DLL.



a.Partage de nœud *b*.Élimination de nœud *c*.Z-élimination *d*. Notation

FIG. 1 – Règles de réductions et notation employée

2 Manipulation d'ensembles de clauses

Les BDD [Bry86, DB98] ont été intensivement utilisés pour décrire des fonctions booléennes. Un BDD est un graphe acyclique dirigé, possédant un unique nœud racine, dont les nœuds sont étiquetés et correspondent soit à une feuille, soit à un nœud interne ayant exactement deux fils. Nous représentons un tel nœud interne n par $\Delta(label, n_1, n_2)$, où (n_1, n_2) correspondent aux fils de n. Les nœuds sont étiquetés par des variables booléennes, et les feuilles par 1 ou 0 .

Selon la sémantique usuelle, chaque nœud est interprété comme une fonction booléenne. Les feuilles 1 et 0 sont respectivement interprétées par *vrai* et *faux* et, si x est une variable booléenne, un nœud $n = \Delta(x, n_1, n_2)$ est interprété comme la fonction *si* x *alors* f_1 *sinon* f_2, où f_1 et f_2 sont les interprétations respectives de n_1 et n_2. Lorsqu'une formule propositionnelle f est représentée par un BDD, chaque chemin de la racine à la feuille 1 correspond à un modèle de f. Étant donné un ordre sur les variables, les OBDD (Ordered BDD) imposent que l'étiquette d'un nœud $n = \Delta(x, n_1, n_2)$ soit plus petite que celles de ses fils n_1 et n_2. Les ROBDD imposent en plus l'élimination de tous les sous-graphes isomorphes. Ceci peut être obtenu par applications successives de la règle de *partage de nœud* (fig. 1-a). La règle *d'élimination de nœud* (fig. 1-b) permet en outre de supprimer tous les nœuds inutiles de la forme $\Delta(x, n, n)$, ce qui traduit que la fonction est indépendante de la valeur de x . Les ZBDD [Min93] utilisent plutôt une autre règle de réduction, la *Z-élimination*, qui revient à remplacer chaque nœud de la forme $\Delta(x, O, n)$ par n (fig. 1-c). Toute variable n'apparaissant pas sur un chemin menant à 1 peut ainsi être implicitement interprétée comme fausse. Les ZBDD se sont avérés efficaces dans le cadre de la représentation d'ensembles à l'aide de leur fonction caractéristique booléenne [Min93].

L'intérêt des ROBDD (que nous appellerons simplement BDD dans la suite) pour coder des formules propositionnelles est de permettre une représentation compacte de très grands ensembles de modèles [eJCM94]. Par contre, ce codage oblige à calculer la forme normale de Shannon de cette formule, opération dont le coût peut parfois être prohibitif (comparativement à SAT, ce problème est au moins aussi difficile que de compter tous les modèles de la formule). Nous proposons ici une autre technique, capable de tirer partie du pouvoir de compression des BDD, mais sans ce défaut majeur. Nous utilisons les ZBDD pour coder l'ensemble des clauses d'une formule sous *fnc*. Les feuilles 1 et 0 restent inchangées mais les nœuds internes sont étiquetés par des littéraux au lieu de variables. Intuitivement, chaque chemin depuis la racine jusqu'à la feuille 1 représente une clause contenant tous les littéraux étiquetés par les nœuds de ce chemin. Étant donné un ordre initial $x_1 < \ldots < x_n$ sur l'ensemble des variables, il est possible d'étendre cet ordre sur les littéraux de la façon suivante : $x_1 < \neg x_1 < \ldots < x_n < \neg x_n$. Plus formellement, la sémantique correspondante peut être décrite par :

- $[\![0]\!] = \emptyset$
- $[\![1]\!] = \{\Box\}$ (l'ensemble ne contenant que la clause vide)
- $[\![\Delta(l, n_1, n_2)]\!] = \{l \vee [\![n_1]\!]\} \cup [\![n_2]\!]$, où $\{l \vee [\![n_1]\!]\}$ représente toutes les clauses obtenues par l'ajout du littéral l aux clauses de $[\![n_1]\!]$

Cette sémantique particulière permet l'introduction de nouvelles règles de réduction. On peut noter que $[\![\Delta(x, \Delta(\neg x, A, B), C)]\!]$ est l'union des trois ensembles de clauses correspondant à $\{x \vee \neg x \vee [\![A]\!]\} \cup \{x \vee [\![B]\!]\} \cup [\![C]\!]$. Comme les clauses contenant à la fois x et $\neg x$ sont des tautologies, elles peuvent être éliminées en appliquant systématiquement la règle d'*élimination des tautologies* : $\Delta(x, \Delta(\neg x, A, B), C) = \Delta(x, B, C)$. Ensuite, on peut noter qu'un nœud de la forme $\Delta(x, A, A)$ correspond à l'ensemble des clauses $(x \vee [\![A]\!]) \cup [\![A]\!]$. Chaque clause du premier ensemble est donc subsumée par une clause du second. La règle d'*élimination de nœuds* peut donc s'appliquer ici, bien que l'on manipule des ZBDD. Cette sémantique permet donc de tirer partie à la fois des règles de réduction des ROBDD et de celles des ZBDD. Comme les littéraux x et $\neg x$ sont ordonnés consécutivement, il peut être pratique d'introduire dans la suite la notation $\nabla(x, A, B, C)$ pour représenter $\Delta(x, A, \Delta(\neg x, B, C))$ ou tout autre réduction de ce nœud (fig. 1-*d*).

2.1 Opérations sur des ensembles de clauses

Minato a montré dans [Min93] comment des opérations de base (union, intersection,...) pouvaient être codées par des opérations sur les ZBDD. Le coût de ces opérations ne dépend que de la taille des ZBDD manipulés. Ici, ce coût n'est donc pas directement lié au nombre de clauses codées. Le premier opérateur exploitant cette nouvelle sémantique est *NoSub*. Il permet simplement d'éliminer toutes les clauses subsumées d'un ensemble de clauses. Sa définition récursive (figure 2) peut être vue comme une reconstruction de bas en haut du ZBDD original. Pour chaque nœud $\Delta(l, A, B)$, la règle de réduction d'*élimination des tautologies* est appliquée si le littéral l est positif, puis les cas de subsomptions dûs au littéral l sont traités. Comme $[\![\Delta(l, A, B)]\!] = \{l \vee ([\![A]\!] \setminus [\![B]\!])\} \cup \{l \vee ([\![A]\!] \cap [\![B]\!])\} \cup [\![B]\!]$, chaque clause de $\{l \vee ([\![A]\!] \cap [\![B]\!])\}$ est subsumée par une clause de B (chacune contient en effet tous les littéraux d'une clause de B plus l). Les clauses subsumées peuvent alors être éliminées en supprimant de A toutes les clauses de B. Les opérateurs *clause-différence* ($\setminus$) et *clause-union* ($\cup$) sont définis de manière similaire (figure 2). Dans leur définition, $l < m$ signifie que l est inférieur à m pour l'ordre des littéraux.

La figure 2 introduit également un nouvel opérateur, appelé *clause-distribution*. Il permet de calculer la distribution, par rapport au connecteur $\vee$, d'un ensemble de clauses sur un autre, en éliminant du résultat toutes les tautologies et les clauses subsumées. À nouveau, la définition de cet opérateur évite de construire des branches inutiles du ZBDD, qui ne contiendraient que des tautologies (comme distribuer A sur E et B sur D dans la définition). Afin d'en faciliter la lecture, la description de cet opérateur utilise la notation ∇. La description complète contient 8 cas différents ($\nabla(x, A, B, C) \times \Delta(\neg x, D, E)$, $\Delta(x, A, B) \times \Delta(\neg x, D, E)$,...). Nous ne donnons ici que le cas le plus général, les autres cas étant similaires. L'opération de *clause-distribution* peut avoir beaucoup d'applications pratiques, allant des démonstrateurs SAT au calcul des impliqués premiers. Dans la suite, nous illustrons son utilisation au cœur d'un démonstrateur SAT basé sur DP.

3 La procédure Davis-Putnam sous un nouvel angle

La procédure originelle de Davis-Putnam [DP60], proposée en 1960, est souvent confondue avec sa version modifiée de 1962 [MDL62]. Les deux versions diffèrent pourtant d'une manière fondamentale. Alors que DLL procède à une recherche avec retour arrière pour SAT, DP élimine les variables une à une par coupure. Ainsi, ce dernier produit une succession de *résolvants*, dont la taille peut croître de manière exponentielle. Dechter et Rish [DR94] ont été les premiers à tenter

$NoSub$:

- $NoSub(1) = 1$ • $NoSub(0) = 0$
- $NoSub(\Delta(l, A, A)) = Nosub(A)$ *(Élimination de nœud)*
- $NoSub(\Delta(l, A, B)) = \Delta(l, NoSub(A) \setminus NoSub(B), NoSub(B))$

clause-différence ($\setminus$) : $(l < m)$

- $0 \setminus A = 0$ • $A \setminus 0 = A$ • $A \setminus A = 0$
- $\Delta(l, A, B) \setminus \Delta(l, C, D) = \Delta(l, (A \setminus C) \setminus D, (B \setminus D))$
- $\Delta(l, A, B) \setminus \Delta(m, C, D) = \Delta(l, A \setminus \Delta(m, C, D), B \setminus \Delta(m, C, D))$
- $\Delta(m, A, B) \setminus \Delta(l, C, D) = \Delta(m, A, B) \setminus D$

clause-union ($\cup$) :

- $A \cup A = A$ • $0 \cup A = A$ • $A \cup B = B \cup A$
- $\Delta(l, A, B) \cup \Delta(l, C, D) = \Delta(l, (A \cup C) \setminus (B \cup D), (B \cup D))$
- $\Delta(l, A, B) \cup \Delta(m, C, D) = \Delta(l, A \setminus (B \cup \Delta(m, C, D)), B \cup \Delta(m, C, D))$

clause-distribution ($\times$) : *(partielle)*

- $\nabla(x, A, B, C) \times \nabla(x, D, E, F) =$
$$\begin{aligned}\nabla(x, (\ (A \times D) \cup (A \times F) \cup (D \times C)\ ) \setminus (C \times F),\\ (\ (B \times E) \cup (B \times F) \cup (E \times C)\ ) \setminus (C \times F),\\ (C \times F))\end{aligned}$$
- $(\dots)$
- $\Delta(l, A, B) \times \Delta(m, C, D) = \Delta(l, (A \times \Delta(m, C, D)) \setminus (B \times \Delta(m, C, D)),$
$$B \times \Delta(m, C, D))$$

FIG. 2 – Définition des opérateurs sur les ensembles de clauses (où $l < m$)

de réhabiliter cette procédure. Ils ont montrés que sa complexité pouvait être bornée par une mesure sur la structure d'un graphe induit par la formule initiale, suivant un ordre donné (*induced width*). Malheureusement, le calcul de cette mesure s'avère être NP-complet, ce qui limite son utilisation en pratique. De plus cette mesure semble borner très supérieurement les performances réelles de DP. Enfin, l'*induced width* ne prend en compte que le nombre potentiel de clauses créés, ce qui n'est plus valable dans le cadre de ZRES, où elles les clauses sont représentées à l'aide de ZBDD.

En 1977, Galil prouva la faiblesse de DP [Gal77] en exhibant une classe particulière de problèmes dont la résolution par DP ne pouvait s'effectuer sans construire une preuve comportant un nombre exponentiel d'étapes de résolution. Dix ans plus tard, Haken [Hak85] puis Urquhart [Urq87], utilisant d'autres classes de problèmes, ont étendu ce résultat à toutes les procédures basées sur la résolution. Ces résultats théoriques suggèrent que ni DP ni DLL ne semblent appropriés pour résoudre de telles instances. Mais, ces résultats s'appuient sur l'hypothèse implicite que les résolutions caractérisant la preuve s'effectuent une par une. Nous montrons ici que l'utilisation des ZBDD, en vue de manipuler directement des ensembles de clauses, peut permettre de reconsidérer ce point. Dans [Gal77], Galil décrit DP comme suit :

I. Choisir une variable propositionnelle x dans f.
II. Remplacer toutes les clauses contenant le littéral x (ou $\neg x$) par celles qui sont obtenues par résolution sur x.
III. a. Si la clause vide apparaît, f est insatisifable.
 b. Si l'ensemble courant est vide, f est satisfiable.
 c. Sinon, répéter les étapes I-III pour le nouvel ensemble de clauses.

Si on factorise dans f les littéraux x et $\neg x$, la formule f peut être écrite comme $(x \vee f_x^+) \wedge (\neg x \vee f_x^-) \wedge f_{x'}$, où f_x^+, f_x^- et $f_{x'}$ sont sous *fnc*. La seconde étape de DP revient alors à réécrire la formule $(f_x^+ \vee f_x^-) \wedge f_{x'}$ sous *fnc*. C'est là que se concentre la réelle difficulté de DP, car la distribution de f_x^+ par rapport à f_x^- peut générer $|f_x^+|.|f_x^-|$ clauses. En terme de ZBDD, cette opération correspond à la *clause-distribution*. Si on note $\lambda(f)$ le ZBDD codant la formule f, le résultat de l'élimination par coupure de la variable x peut être obtenu grâce aux opérateurs précédemment définis :

$$\textit{élimination-coupure}(f, x) \;=\; (\,\lambda(f_x^+) \;\times\; \lambda(f_x^-)\,) \;\cup\; \lambda(f_{x'})$$

Le coût de cette opération ne dépend pas du nombre de clauses, mais seulement du nombre de nœuds de $\lambda(f_x^+)$, $\lambda(f_x^-)$ et de $\lambda(f_{x'})$. Il est donc possible d'éliminer par coupure une variable mettant en jeu un nombre important de clauses, si les ZBDD correspondants sont de taille raisonnable. ***L'utilisation d'une telle structure de données, associées à l'opérateur de* clause-distribution *conduit donc à remettre en cause le consensus sur l'inefficacité de DP***.

Pour évaluer l'efficacité de cette approche, le système ZRES[1], a été implémenté. Cette nouvelle procédure de DP est écrite en C et s'appuie sur le paquetage *Cudd* [Som], qui fournit les opérations standards sur les ZBDD ainsi que de précieuses fonctions de réordonnancement dynamique. Deux versions ont été réalisées. La version standard (*std*) utilise les opérateurs de distributions standard sur les ZBDD, puis élimine les tautologies et les clauses subsumées. La version *cd* s'appuie sur l'opérateur de *clause-distribution* pour effectuer ces trois étapes simultanément.

4 Résultats expérimentaux

Tous les tests ont été exécutés sur un Pentium-II 400MHz[2] sous Linux avec 256Mo. Nos résultats sont comparés avec ceux de deux implémentations de DLL : ASAT [Dub], qui est une implémentation simple mais efficace de DLL et SATO 3.0 [Zha97], qui incorpore de récentes optimisations, telles que le *back-jumping* ou la mémorisation des conflits. À notre connaissance, SATO est le système qui obtient les meilleurs résultats sur les exemples de Dimacs [Dim]. Toutes les valeurs correspondent aux temps CPU exprimés en secondes. Lorsqu'un exemple n'a pu être résolu en moins de 10000s, il est comptabilisé pour 10000s.

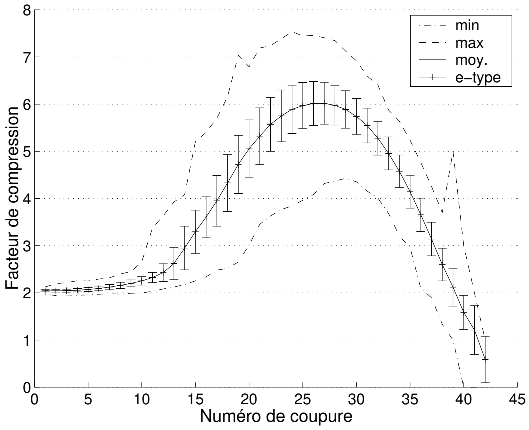
4.1 Compression de clauses par ZBDD

Le choix des ZBBD repose sur l'hypothèse selon laquelle les instances structurées devraient présenter des régularités. Pour valider le bien-fondé de cette supposition, nous avons étudié sur différents exemples la variation du *taux de compression*, que l'on peut définir comme le rapport entre le nombre de littéraux d'une formule et le nombre de nœuds du ZBDD codant cette formule.

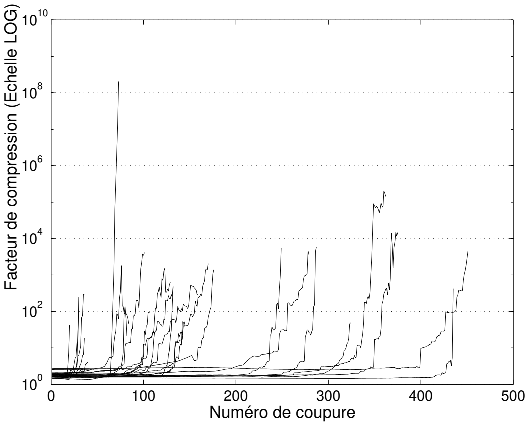
Dans un premier temps, nous avons mesuré l'évolution de ce taux durant la résolution d'instances aléatoires 3-SAT, un problème de référence dans la littérature [MSL92]. Nous avons testé 1000 instances, de 180 clauses et 42 variables. Les résultats (3.*a*) mettent en évidence qu'au début du calcul, le taux est pratiquement le même pour toutes les instances : 2.05 (écart-type 0.02). Ensuite, ce taux croît jusqu'à un maximum de 6.15 (0.44) pour finalement redescendre. On peut observer que la valeur maximale correspond au moment où ZRES manipule le plus grand nombre de clauses. Il décroit ensuite parallèlement au nombre de clauses (l'évolution détaillée du nombre

[1] ZRES est disponible sur http ://www.lri.fr/~simon/research/zres.

[2] L'indicateur de comparaison entre machine donné par [Dim] donne un gain de temps constant de 305%



a.Aléatoire 180 clauses (1000 exp. au seuil)



b. Sur Aralia

c. Sur ii8

Problèmes	Entrée	Meilleur
Pret150	1.88 (*0*)	1157 (*0*)
Par32-c	2.14 (*0.02*)	10^7 (*10^7*)
Massacci-R4	7.16 (*0.25*)	72.9 (*8.95*)
Flat-50	1.76 (*0.01*)	9.61 (*2.14*)
Ssa	1.72 (*0.01*)	5.72 (*0.28*)

d. Résumé

FIG. 3 – Évolution et résumé des facteurs de compression

de clauses peut être observée dans [CS00]). Il peut paraître surprenant d'obtenir un taux de compression supérieur à 1 sur des exemples aléatoires. Le taux initial traduit simplement le fait que, compte tenu des caractéristiques des instances générées (au seuil), il y a statistiquement des chances de pouvoir factoriser le début ou la fin d'un certain nombre de clauses. Le fait que ce taux augmente au cours du calcul s'explique par les coupures successives, induisant la duplication de portions de clauses, qui font que les résolvants successifs ne présentent plus les caractéristiques d'une formule aléatoire. Au contraire, ils comportent de nombreuses redondances, qui sont efficacement prises en compte par le ZBDD.

Observons à présent le comportement de ZRES sur quelques instances structurées. La base de tests d'Aralia [Rau] n'est habituellement pas utilisée par des démonstrateurs SAT. Ces exemples modélisent des systèmes physiques et correspondent à des problèmes consistant à n'éliminer par coupure qu'un sous ensemble des variables de départ (la fin du calcul correspond au calcul d'une formule). La figure 3-*b* représente la trace de ZRES sur cette base de test (une instance par courbe). On peut observer sur la même figure que les taux de compression pour ces benchmarks sont bien supérieurs à ceux obtenus pour les exemples aléatoires, tout au moins en fin de calcul. L'aptitude des ZBDD à capturer les régularités de ces instances est donc manifeste (avec un

TAB. 1 – Temps CPU (s) sur quelques instances de Hole-n

Instances	ASAT	SATO	ZRES *std*	ZRES *cd*
Hole-09	11.94	8.90	1.87	1.01
Hole-10	141.96	80.94	3.26	1.61
Hole-11	1960.77	7373.65	5.76	2.65
Hole-12	10000	10000	10.18	4.06
Hole-20	–	–	654.8	69
Hole-30	–	–	10000	1102
Hole-40	–	–	–	9421

rapport pouvant dépasser 10^8 littéraux par nœud). La figure 3.c détaille l'évolution des taux de compression sur certains exemples de la famille $ii8$ issue de la base de benchmarks de Dimacs. À nouveau, on peut observer une différence d'échelle, par rapport aux résultats obtenus sur les instances aléatoires. Le tableau 3-*d* complète ces résultats, en précisant les taux de compression obtenus sur d'autres instances de Dimacs, ainsi que sur le problème proposé par Massacci [MM00], qui correspond à la résolution d'un problème de cryptographie. Notons que ce dernier, ainsi que Parity-32, n'ont cependant pas pu être entièrement résolus par ZRES. On remarque par contre que, pour ssa et $flat$, le taux de compression maximum n'est pas très important. Cependant, bien que correspondant à des problèmes concrets (coloration de graphes et tests de circuits), les instances de ces deux problèmes semblent avoir une forte composante aléatoire. Ceci pourrait expliquer les résultats obtenus. ***Les résultats expérimentaux sur l'évolution du facteur de compression confirment l'aptitude des ZBDD à capturer les régularités présentes dans les exemples.***

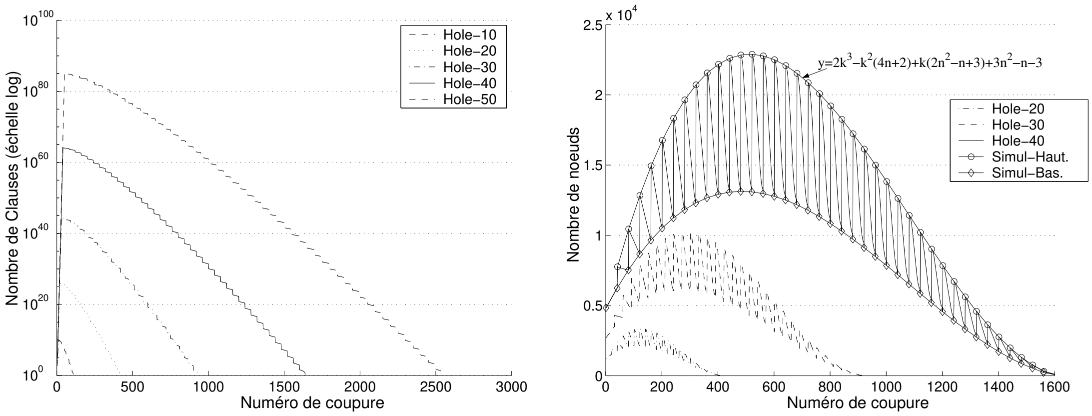
4.2 Le problème des Pigeons

Le célèbre problème des Pigeons a été l'objet de beaucoup d'attention depuis son utilisation par Haken [Hak85] pour démontrer la faiblesse de la résolution. Le problème Hole-n traduit simplement que $n + 1$ pigeons ne peuvent tenir dans n trous. L'apparente simplicité de ce problème contraste avec la difficulté à le traiter par résolution. Ceci a certainement contribué pour beaucoup à sa célébrité. Différentes heuristiques ont été testées. Il est surprenant de constater que les critères de choix de variable avec des structures de ZBDD ne coïncident pas forcément avec ceux utilisés avec des structures plus classiques. Alors que d'ordinaire il semble assez naturel de choisir la variable dont la coupure génère le moins de clauses, les meilleurs résultats, sur ce problème, ont été obtenus avec une heuristique tendant à maximiser le nombre de clauses produites à chaque coupure. Les résultats obtenus sont donnés table 1.

La première observation qui peut être faite est que, non seulement ZRES *cd* surpasse nettement ASAT et SATO, mais il peut aussi résoudre des problèmes de taille beaucoup plus grande. ZRES est même parvenu à résoudre Hole-55 (84756 clauses) en moins de 2 jours. En comparaison, Hole-10 ne contient que 561 clauses[3]. La seconde observation concerne les deux versions de ZRES. Les résultats obtenus confirment, sur ce problème, l'efficacité de l'opérateur *clause-distribution*. Les gains obtenus semblent même augmenter avec la taille du problème traité.

En observant l'évolution du nombre de clauses sur la figure 4-*a*, on peut remarquer que deux phases se distinguent. La première correspond aux n premières coupures et introduit une crois-

[3] Hole-n contiens $n * (n + 1)$ variables et $\frac{1}{2}(n^3 + n^2) + n + 1$ clauses.



*a.*Nombre de clauses *b.*Nombre de nœuds

FIG. 4 – Comportement de ZRES sur le problème des Pigeons

sance clairement exponentielle. En étudiant précisément la liste des coupures effectuées, on peut montrer qu'à la fin de la première phase le nombre total de clauses est de n^n. La seconde phase correspond à une décroissance par paliers successifs, chacun des n paliers étant de largeur $n+1$. Pour mieux comprendre le comportement de ZRES sur ce problème, on peut observer attentivement l'évolution du nombre total de clauses (figure 4-*a*) ainsi que celle du nombre total de nœuds (figure 4-*b*) après chaque coupure. À la fin de la première phase, le nombre de clauses est réellement impressionnant mais dans le même temps, le nombre de nœuds n'augmente que de 10% (noter la différence d'échelle entre les deux figures). La seconde phase est caractérisée par une courbe oscillant entre deux courbes, correspondant respectivemement aux valeurs minimales et maximales de chaque période (une période correspondant à $n+1$ coupures). Durant la première coupure de chaque période, le nombre de nœuds passe immédiatement de sa valeur minimale à sa valeur maximale. Pendantles n coupures suivantes, il décroît régulièrement, jusqu'à la valeur minimale de la période suivante. Ces oscillations suggèrent clairement la répétition d'un même phénomène, étroitement lié à la structure de la formule. Les brusques accroissements du nombre de nœuds correspondent à l'élimination de variables, dont la suppression introduit des dissymétries et affaiblit le niveau de compression, alors que les coupures suivantes de la même période restaurent progressivement les régularités.

La forme de l'enveloppe passant par les valeurs maximales durant chaque période contraste avec la variation du nombre total de clauses. Cette première enveloppe croît progressivement jusqu'à atteindre sa valeur maximale après environ $\frac{n^2}{3}$ coupures. Nous avons conduit une étude analytique sur la structure précise des ZBDD successifs, afin d'évaluer les valeurs minimales et maximales. Les résultats obtenus confirment les valeurs expérimentales et prouvent que les enveloppes *min* et *max* correspondent à des fonctions polynômiales de degré 3. Nous représentons, sur la figure 4-*b*, pour le cas Hole-40, les courbes correspondant aux enveloppes *max* et *min*, et donnons l'équation précise caractérisant l'enveloppe du haut, qui doit être lue avec $k = \lfloor \frac{x}{n} \rfloor$. ***En conclusion, les résultats expérimentaux et théoriques montrent que pour le problème des Pigeons, la taille des résolvants intermédiaires croît de manière exponentielle en nombre de clauses, mais polynômialement en nombre de nœuds.* ZRES *semble donc pouvoir résoudre efficacement ce problème, malgré les mauvais résultats théoriques caractérisant DP.***

TAB. 2 – Temps CPU total (s) sur 100 instances de Urq-3

Système	Temps CPU Total	# Résolus	Temps CPU Moy. (résolus)
ASAT	404 287	69	1366
SATO	776 364	26	1398
ZRES *cd*	69.2	100	0.69

TAB. 3 – Moyenne (écart-type) sur 1000 instances de Urquhart.

Urq	Nb. Var. Initial .	Nb. Cl. Initial	Nb. Cl. Max	Nb. Nœuds Max	Temps de ZRES
4	78 *(5.8)*	714 *(166)*	10^{10} *(10^1)*	440 *(46)*	1.72 *(0.2)*
6	178 *(8.7)*	1683 *(261)*	10^{24} *(10^2)*	1011 *(69)*	8.88 *(0.9)*
8	318 *(11.3)*	3024 *(348)*	10^{43} *(10^3)*	1808 *(90)*	29.6 *(2.2)*
10	498 *(13.9)*	4754 *(433)*	10^{67} *(10^4)*	2833 *(110)*	72.0 *(4.3)*

4.3 Le problème de Urquhart

Le problème de Urquhart [Urq87] caractérise une propriété des *expander graphs*, dans une version légèrement différente de celle proposée initialement par Galil [Gal77] pour établir une borne inférieure exponentielle au coût de la résolution dirigée. Chaque instance de Urquhart correspond à un ensemble de sous-bases. Les clauses d'une même sous-base ont toutes la même longueur et sont construites sur les même variables propositionnelles. Elles ne diffèrent que dans le choix des littéraux positifs ou négatifs. Une contrainte additionnelle impose que toutes les clauses d'une même sous-base aient un nombre pair ou impair de littéraux négatifs.

Les résultats obtenus par ZRES sur ces exemples (table 2) sont encore une fois bien meilleurs que ceux des deux autres implémentations de DLL testées. L'heuristique utilisée ici est l'heuristique classique, qui tend à minimiser le nombre de clauses produites. Comme pour le problème des pigeons, de plus grandes instances du problème de Urquhart ont été générées, afin d'observer l'évolution du comportement de ZRES. Les résultats montrent clairement que le nombre maximum de clauses pour chaque calcul est exponentiellement plus grand que le nombre maximum de nœuds atteint (table 3). Notons que nous ne donnons pas ici le temps de ZRES *std* car cette classe de problème ne permet pas d'observer de différence significative entre les deux versions.

Donnons maintenant une intuition aidant à comprendre ce bon comportement. Par construction, chaque sous-base construite sur i variables contient 2^{i-1} clauses. En terme de ZBDD, cet ensemble de clauses hautement structuré peut être représenté en seulement $O(i)$ nœuds. Les coupures successives amplifient encore plus cette différence exponentielle, une distribution entre deux de ces sous-ensembles disjoints, construits avec i (resp. j) variables, conduit en effet à produire 2^{i+j-2} clauses qui peuvent être représentées en seulement $O(i+j)$ nœuds. ***Comme pour le problème des Pigeons, la résolution des instances de Urquhart conduit à produire un nombre exponentiel de clauses, représentées par un ZBDD de taille polynômiale. Ce second problème reste néanmoins traitable par* ZRES.**

4.4 Impact de l'opérateur de *clause-distribution*

L'intérêt de l'opérateur de *clause-distribution* est manifeste dans le cas du problème des Pigeons. Il est surprenant de constater qu'on n'observe pas de gains similaires avec le problème

TAB. 4 – Résultats sur 1000 instances 3-SAT avec 190 Cl. et 44 Var.

ZRES	Temps CPU	Nb. Nœuds Max.	Taille Mémoire (Mo)
std	306 *(277)*	5 638 848 *(3 973 840)*	186 *(102)*
cd	118 *(101)*	284 474 *(134 324)*	13 *(2)*

de Urquhart. Nous pensons que ce dernier problème est tellement compressible que la *clause-distribution* y a un impact très réduit. De plus, sur de telles instances, peu de clauses sont éliminées par subsomption. Le gain induit par l'élimination au vol de telles clauses est donc marginal. Pour évaluer l'effet de cet opérateur dans un contexte plus général, nous proposons une comparaison expérimentale entre ZRES *std* et ZRES *cd* sur des instances 3-SAT aléatoires.

Le travail de [CS00] avait déjà mis en évidence que DP n'était pas adéquat pour les instances aléatoires. Les résultats obtenus (cf table 4) confirment ce point et montrent que ZRES n'apporte pas d'améliorations significatives pour de telles instances sans structure. En raison de leur manque de régularité, les instances aléatoires restent faiblement compressibles. Même si le taux de compression peut augmenter (*cf* section 4.1), le nombre de nœuds manipulés augmente trop rapidement pour traiter ces exemples aussi efficacement que les approches de type DLL. Notons en effet que les résultats de ASAT et SATO ne sont pas mentionnés ici car de telles instances sont résolues instantanément par ces deux systèmes. En ce qui concerne la comparaison des deux versions de ZRES, on peut constater une différence significative, notamment en terme de consommation mémoire. L'explication peut se retrouver dans [CS00], où les auteurs ont montré l'importance des tests de subsomption pour de tels exemples.

5 Conclusion

L'établissement de bornes inférieures exponentielles sur la longueur des preuves de certains exemples a conduit à considérer les procédures basées sur la résolution comme intrinsèquement limitées. Il est souvent suggéré que, pour dépasser ces limites qui apparaissent même sur des problèmes simples comme celui des Pigeons, il est nécessaire de recourir à des systèmes de preuve plus puissants. Notre sentiment est que, dans certains cas, la longueur d'une preuve n'est pas un bon indicateur de sa complexité, tout comme la longueur d'un message n'est pas significatif en théorie de l'information. Nous avons montré qu'en utilisant des opérations spécialisées pour la manipulation d'ensembles de clauses codés par des ZBDD, la procédure originelle de DP pouvait raisonner sur un nombre exponentiel de clauses, tout en manipulant des structures de données de taille polynômiales. Ainsi, bien qu'habituellement perçue comme inefficace, la procédure de DP est capable de traiter en pratique des exemples difficiles pour la résolution.

Plusieurs directions semblent intéressantes pour la suite de ce travail. L'opérateur de *clause-distribution*, bâti sur les ZBDD, a été utilisé au cœur d'un démonstrateur SAT, ce qui permet de considérer l'algorithme DP sous un nouvel angle. De nouvelles heuristiques, basées sur la structure des ZBDD plutôt que sur le nombre de clauses, peuvent être développées. Nous pensons que l'opérateur de *clause-distribution* peut aussi être utilisé efficacement dans d'autres domaines, comme la génération des impliqués/impliquants premiers. Des travaux préliminaires dans ce sens semblent prometteurs. Du point de vue du problème SAT, nous pensons que la procédure DLL peut également profiter de la représentation sous forme de ZBDD pour manipuler de grands ensembles de clauses de manière compacte. Une autre perspective pourrait être de développer un algorithme hybride, mêlant DP et DLL, et tirant parti du meilleur des deux approches.

Références

[Bry86] R.E. Bryant. Graph - based algorithms for boolean function manipulation. *IEEE Trans. on Comp.*, 35(8) :677–691, 1986.

[CS00] Ph. Chatalic and L. Simon. Davis and putnam 40 years later : a first experimentation. Technical Report 1237, LRI, Orsay, France, 2000.

[DB98] R. Drechsler and B. Becker. *Binary Decision Diagram : Theory and Implementation*. Kluwer Academic Publisher, 1998.

[Dim] The dimacs challenge benchmarks. ftp ://ftp.rutgers.dimacs.edu/challenges/sat.

[dK92] J. de Kleer. An improved incremental algorithm for generating prime implicates. In *AAAI'92*, pages 780–785, 1992.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, pages 201–215, 1960.

[DR94] R. Dechter and I. Rish. Directional resolution : The davis-putnam procedure, revisited. In *Proceedings of KR-94*, pages 134–145, 1994.

[Dub] O. Dubois. Can a very simple algorithm be efficient for sat ? ftp ://ftp.dimacs.rutgers.edu/pub/challenges/sat/contributed/dubois.

[eJCM94] O. Coudert et J-C. Madre. Une approche intentionnelle du calcul des implicants premiers et essentiels des fonctions booléennes. *Informatique théorique et Applications*, 1994.

[Gal77] Z. Galil. On the complexity of regular resolution and the davis-putnam procedure. *Theorical Computer Science*, 4 :23–46, 1977.

[Hak85] A. Haken. The intractability of resolution. *Theorical Computer Science*, 39 :297–308, 1985.

[MDL62] G. Logeman M. Davis and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, pages 394–397, 1962.

[Min93] S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *30th ACM/IEEE Design Automation Conference*, 1993.

[MM00] F. Massacci and L. Marraro. Logical cryptanalysis as a sat-problem : Encoding and analysis of the u.s. data encryption standard. *JAR*, 2000.

[MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. *AAAI'92*, pages 459–465, 1992.

[Rau] A. Rauzy. Boolean models for reliability analysis : a benchmark. Technical report, LaBRI - Université Bordeaux I.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI-92*, pages 440–446, 1992.

[Som] F. Somenzy. Cudd release 2.3.0. http ://bessie.colorado.edu/˜fabio.

[Urq87] A. Urquhart. Hard examples for resolution. *J. of the ACM*, 34 :209–219, 1987.

[Zha97] H. Zhang. Sato : An efficient propositional prover. In *CADE-14*, LNCS 1249, pages 272–275, 1997.