Multi-Resolution on Compressed Sets of Clauses

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

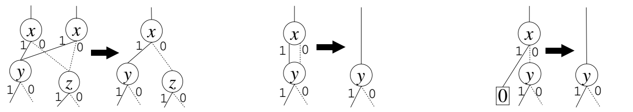
Abstract

This paper presents a system based on new operators for handling sets of propositional clauses represented by means of ZBDDs. The high compression power of such data structures allows efficient encodings of structured instances. A specialized operator for the distribution of sets of clauses is introduced and used for performing multi-resolution on clause sets. Cut eliminations between sets of clauses of exponential size may then be performed using polynomial size data structures. The ZRES *system, a new implementation of the Davis-Putnam procedure of 1960, solves two hard problems for resolution, that are currently out of the scope of the best SAT provers.*

1 Introduction

Recent years have seen a lot of work using propositional logic as a framework for knowledge representation and problem solving. The poor expressive power of propositional logic is counterbalanced by its simplicity which allows simple, but efficient, provers to be constructed. In particular, the SAT problem has been the focus of much interest, ranging from refinement and optimization of complete procedures [22] to the introduction of new incomplete methods [17]. Hard random instances have been identified [14] and reference problems have been gathered in benchmarks, subject to program competitions (e.g. [7, 18]). If its central place in complexity theory is probably part of the motivation of such work, the resolution of real world problems is also an attractive challenge for SAT. In the area of complete methods, most current SAT provers are based on the Davis, Logeman and Loveland procedure (DLL) [11]. One reason of success of such solvers is their ability to limit memory usage. This is due to the fact that they essentially make choices, resulting in simplifications and unit propagations. The difficulty comes from the number of possible choices which is exponential. The role of heuristics is then central. However, heuristics which are efficient on random problems are seldom appropriate for structured instances, on which dynamic heuristics based on learning techniques give better results [22]. Other problems, such as prime implicants/implicates generation or knowledge base compilation, often handle large sets of clauses. Then, the difficulty rather comes from the size of such sets that may grow exponentially. Whatever the goal, one is rapidly faced with combinatorial explosion. On structured instances, the only way to deal with such growth seems to be to take the structure of the available information into account. If heuristics can prune the search space efficiently, savings in space may be achieved by means of *compression* algorithms.

This paper focuses on data structures used for encoding sets of clauses and on associated operators. We are particularly interested in instances with a special structure. Our feeling is that such instances should show some regularities through the encoding. An appropriate data structure should be able to take advantage of such regularities to handle a more compact representation of the encoded formula. For cnf formulæ, such regularities may correspond to subsets of literals that appear in many different clauses. We propose to use a data structure allowing the factorization of common subsets of literals. *Tries* structures [5] enable the factorization of clauses beginning with the same sequence of literals. They remain the state-of-the-art data structures for subsumption checking [22]. We further generalize this idea to factorize simultaneously ends of clauses. In practice, we use a variant of BDDs [2], called ZBDDs [13], to store sets of clauses by means of their characteristic functions. We show that large sets of clauses corresponding to structured instances may be efficiently compressed in that way. Moreover, such structures are also well suited to subsumption checking, and all set operations can be realized as operations on ZBDDs. We introduce a new operator for performing multiple resolutions on sets of clauses represented by ZBDDs, in a single step. We use this operator in an implementation of the original Davis and Putnam algorithm [4] (DP as opposed to DLL).



a.Node sharing *b*.Node elimination *c*.Z-elimination

Figure 1. Reductions types in BDDs

The next section briefly recalls the basic principles of BDDs and ZBDDs structures. It introduces our technique to encode sets of clauses by means of ZBDDs, as well as new operators. In the third section we show that expressing cut-elimination using these operators makes it possible to perform multi-resolution on sets of clauses. We illustrate its use in a new implementation of DP, called ZRES. The last sections are devoted to empirical studies. Two classes of hard instances for resolution, namely the Pigeon Hole problem and the Urquhart problem, are tested and compression capabilities of ZBDDs are evaluated.

2 Representing sets of clauses with BDDs

2.1 Encoding boolean functions with BDDs

A BDD [2, 16] is a directed acyclic graph with labeled nodes, a unique source node, and such that each node is either a sink node or an internal node (denoted by $\Delta(x, n_1, n_2)$) having two children (n_1, n_2) and a *label* x. BDDs encoding boolean functions have only two sinks, labeled 1 and 0, and their internal nodes are labeled with boolean variables $\{x, y, \dots\}$. The classical semantics interprets sinks 1 and 0 as *true* and *false*, and any internal node $n = \Delta(x, n_1, n_2)$ as the function f = *if* x *then* f_1 *else* f_2, where f_1 and f_2 are the respective interpretations of n_1 and n_2. Each path from the source node to the sink 1 of a BDD thus corresponds to a model of the encoded formula. Given an ordering over variables, Ordered BDDs (OBDDs) require the label of any node to be smaller than the labels of its children. OBDDs can thus be viewed as binary trees encoding the Shannon decomposition [16] of the initial formula. To optimize memory usage, additional reduction rules may be used. Reduced OBDDs (ROBDDs) require the graph not to contain any isomorphic subgraphs (*node sharing* rule, fig 1-a) or any useless node of the form $\Delta(x, n, n)$, that do not care about its label value (*node elimination* rule, fig 1-b). ZBDDs [13] replace *node elimination* by *Z-elimination* (fig. 1-c), for which useless nodes are those of the form $\Delta(x, 0, n)$ which default interpretation is *false*. In the following, without further precisions, we simply use the term BDD in place of ROBDD.

The main advantage of using a BDD for encoding a propositional formula is that it can describe very large sets of models in a very compact way. However this requires computing the Shannon normal form of the formula, which may be very expensive if the formula is in conjunctive normal form (cnf). From the SAT point of view, since the BDD encodes the set of all models, this is as difficult as counting them, which is #P-complete.

2.2 Encoding sets of clauses with ZBDDs

To benefit from the high compression capabilities of BDDs while avoiding the computation of Shannon normal forms, we propose an alternative encoding and use a ZBDD to encode the set of clauses of a cnf formula. We still use two sink nodes 1 and 0 but internal nodes are labeled with literals (instead of variables). Intuitively, each path, from the source node to the 1 sink, represents a clause containing all the literals labeling the parent nodes of 1-arcs of this path. One may notice that in [1, 16], ZBDDs are also used as data structures for encoding sequences of literals corresponding to sets of prime implicants. However all these approaches first compute the ROBDD of the initial formula, and use a property of the Shannon decomposition (the *decomposition theorem*) to derive the set of prime implicants. Our approach is different since the ZBDD is directly constructed from the original cnf. In the following, we assume that, given an initial ordering $x_1 < \dots < x_n$ on the set of variables, we use for such ZBDDs the extended literal ordering corresponding to $x_1 < \neg x_1 < \dots < x_n < \neg x_n$. Given a set of clauses S, we denote by f_S the formula corresponding to the conjunction of all clauses of S. Encoding a set of clauses via a ZBDD essentially helps to factorize common beginnings and ends of clauses. For instance, with the initial variable ordering $x < y < z$, the set $S_1 = \{x \vee y \vee \neg z, \neg x \vee \neg y, \neg y \vee \neg z, z\}$ is represented[1] by the ZBDD of figure 2.*a*.

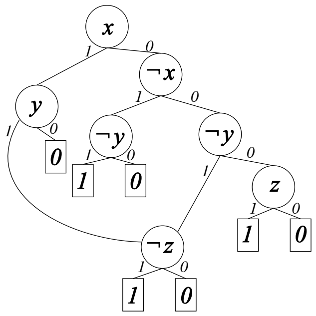
The semantics we use interprets such ZBDDs as sets of clauses. This may be formalized by:

- $[\![0]\!] = \emptyset$ (the empty set of clause)
- $[\![1]\!] = \{\Box\}$ (the set reduced to the empty clause)
- $[\![\Delta(l, n_1, n_2)]\!] = \{l \vee [\![n_1]\!]\} \cup [\![n_2]\!]$, where $\{l \vee [\![n_1]\!]\}$ denotes the set of clauses obtained by adding the literal l to each of the clauses of $[\![n_1]\!]$.

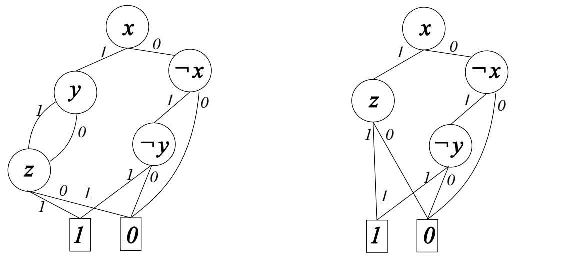
Definition 1 *Let* $nc(\Delta)$ *(resp.* $nl(\Delta)$*) be defined by:*
• $nc(0)$=0, $nc(1)$=1, $nc(\Delta(l, A, B))$=$nc(A)$+$nc(B)$
• $nl(0)$=0, $nl(1)$=0, $nl(\Delta(l, A, B))$=$nc(A)$+$nl(A)$+$nl(B)$

With our semantics, it can be easily shown that $nc(\Delta)$ (resp. $nl(\Delta)$) compute respectively the number of clauses

[1] Sink nodes 1 and 0 have been duplicated for a better readability.



a. ZBDD encoding of S_1



b. S_2 before node elimination *c*. after node elimination

Figure 2. ZBDD encoding of sets of clauses

and the number of literals of $\llbracket\Delta\rrbracket$. This properties can be proven using the Z-elimination rule semantics.

Theorem 1 *Let C be a set of clauses corresponding to a formula f_C, the number of internal nodes of the ZBDD Δ encoding f_C is always at most equal to the total number of literals of C, $nl(\Delta)$.*

Interpreting ZBDDs as sets of clauses has a significant impact on the encoding, one of which is the easy detection of some kinds of subsumed clauses. Let us consider the set $S_2 = \{\neg x \vee \neg y, x \vee y \vee z, x \vee z, \neg x \vee \neg y \vee z\}$. The corresponding ZBDD is represented on figure 2-b. Note that the clause $\neg x \vee \neg y \vee z$, which is subsumed by the clause $\neg x \vee \neg y$, cannot be explicitely represented. More generally, any clause c_1 of length $n + k$, such that the n first literals correspond to another clause c_2 cannot be explicitly represented. But, since c_1 is subsumed by c_2, it can be simply removed. Another simplification may also be performed on the node labelled by y. The path going through its 1-arc corresponds to the clause $x \vee y \vee z$ while the path going through its 0-arc corresponds to the clause $x \vee z$, which clearly subsumes the previous one. The ZBDD obtained after elimination of the y node (fig. 2-c) thus corresponds to an equivalent set of clauses. More generally, any node of the form $\Delta(x, A, A)$ corresponds to the set of clauses $\{x \vee \llbracket A\rrbracket\} \cup \llbracket A\rrbracket$. So, each clause of the first set is subsumed by a clause of the second set. With our semantics, it is thus possible to take advantage of both the Z-elimination rule and the ROBDD node-elimination rule. Another easy simplification is the elimination of tautologies. A ZBDD of the form $\Delta(x, \Delta(\neg x, A, B), C)$ corresponds to the union of three sets of clauses : $\{x \vee \{\neg x \vee \llbracket A\rrbracket\}\} \cup \{x \vee \llbracket B\rrbracket\} \cup \llbracket C\rrbracket$. However, clauses containing both x and $\neg x$ are tautologies. A simple way to eliminate them is to apply the *tautology elimination rule*, which systematically replaces nodes of the form $\Delta(x, \Delta(\neg x, A, B), C)$ by $\Delta(x, B, C)$. In the following, we assume that all considered ZBDDs are constructed by systematically applying the appropriate reduction rules.

2.3 Operations on clause sets

Minato has shown in [13] that basic set operations (union, intersection,...) may be realized as operations on ZBDDs. We here introduce new recursive operators that take advantage of our special semantics, but that could not be used with usual semantics of ZBDDs. In the following, literals are denoted by $\{l, m, \ldots\}$. Proofs of most properties are done by induction on the sum of the size (nl) of the sets of clauses corresponding to operators' arguments[2]. We use the following lemma:

Lemma 1 *Let $A = \Delta(l, A_1, A_2)$ and $B = \Delta(m, B_1, B_2)$ be two ZBDDs. If $l < m$, then no clause of $\llbracket B\rrbracket$ contains the literal l and no clause of $\{l \vee \llbracket A_1\rrbracket\}$ subsumes a clause of $\llbracket A_2\rrbracket$.*

We now introduce a first binary operator on ZBDD, that is used in the definition of the next operators:

Definition 2 (Subsumed difference) *The $\mathbin{\backslash\!\!\backslash}$ operator is defined by:*

$T_1 :\ 0 \mathbin{\backslash\!\!\backslash} A = 0$

T_{2a}: $1 \mathbin{\backslash\!\!\backslash} 1 = 0$ $\qquad$ T_{2b}: *if* $A \neq 1$, $1 \mathbin{\backslash\!\!\backslash} A = 1$

T_{3a}: $\Delta(l, A_1, A_2) \mathbin{\backslash\!\!\backslash} 0 = \Delta(l, A_1, A_2)$

T_{3b}: $\Delta(l, A_1, A_2) \mathbin{\backslash\!\!\backslash} 1 = 0$

$R_1 :\ (l > m)\ \Delta(l, A_1, A_2) \mathbin{\backslash\!\!\backslash} \Delta(m, B_1, B_2) = \Delta(l, A_1, A_2) \mathbin{\backslash\!\!\backslash} B_2$

$R_2 :\ (l < m)\ \Delta(l, A_1, A_2) \mathbin{\backslash\!\!\backslash} \Delta(m, B_1, B_2) = \Delta(l,\ A_1 \mathbin{\backslash\!\!\backslash} \Delta(m, B_1, B_2),\ A_2 \mathbin{\backslash\!\!\backslash} \Delta(m, B_1, B_2))$

$R_3 :\ \Delta(l, A_1, A_2) \mathbin{\backslash\!\!\backslash} \Delta(l, B_1, B_2) = \Delta(l,\ (A_1 \mathbin{\backslash\!\!\backslash} B_1) \mathbin{\backslash\!\!\backslash} B_2,\ (A_2 \mathbin{\backslash\!\!\backslash} B_2))$

Theorem 2 *Given two sets of clauses S_1 and S_2, let us denote by $S_1 \mathbin{\widehat{\backslash\!\!\backslash}} S_2$ the set of clauses obtained by removing from S_1 all the clauses that are subsumed by some clause of S_2. Let A and B be two ZBDDs encoding sets of clauses, then $\llbracket A \mathbin{\backslash\!\!\backslash} B\rrbracket = \llbracket A\rrbracket \mathbin{\widehat{\backslash\!\!\backslash}} \llbracket B\rrbracket$.*

[2] Full proofs are available from the authors.

SKETCH OF PROOF: For the rule R_1, since $l > m$, by lemma 1, no clause of $\{m \vee [\![B_1]\!]\}$ may subsume a clause of $[\![A]\!]$ and thus $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = [\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!]$. For the rule R_2, since $l < m$, by lemma 1, no clause of $[\![B]\!]$ does contains the literal l. Thus if it subsumes some clause of $[\![A]\!]$ it necessarily subsumes the same clause without the literal l in $[\![A_1]\!]$. For the rule R_3, we first prove that $nl(A \setminus\!\!\setminus B) \leq nl(A)$ to use our induction hypothesis twice (to compute $(A_1 \setminus\!\!\setminus B_1) \setminus\!\!\setminus B_2$). Then, clauses of $[\![B]\!]$ containing l may only subsume clauses of $[\![A]\!]$ containing l. But the clauses of $[\![B]\!]$ not containing l may subsume a clause of either $[\![A_1]\!]$ or $[\![A_2]\!]$.

Corollary 1 *If A and B encode two sets of clauses free of subsumptions, then $A \setminus\!\!\setminus B$ is also free of subsumptions.*

Corollary 2 *If $A = \Delta(l, A_1, A_2)$ is such that A_1 and A_2 are free of subsumptions, then $\Delta(l, A_1 \setminus\!\!\setminus A_2, A_2)$ is also free of subsumptions.*

The purpose of our second operator is to remove subsumed clauses from a given set of clauses.

Definition 3 (Subsumption elimination: $NoSub$)

T_1: $NoSub(1) = 1$ $\qquad$ T_2: $NoSub(0) = 0$

R_1: $NoSub(\Delta(l, A, A)) = Nosub(A)$
(node-elimination)

R_2: *if* $A_1 \neq A_2$, $NoSub(\Delta(l, A_1, A_2)) =$
$\Delta(l,\ NoSub(A_1) \setminus\!\!\setminus NoSub(A_2),\ NoSub(A_2))$

Theorem 3 *Let A be a ZBDD encoding a set of clauses, $[\![NoSub(A)]\!]$ and $[\![A]\!]$ are logically equivalent sets of clauses ($f_{[\![A]\!]} \leftrightarrow f_{[\![NoSub(A)]\!]}$) and $[\![NoSub(A)]\!]$ does not contain any subsumed clause.*

The proof is essentially based on lemma 1 and corollary 2. We now introduce a binary operator that performs the union of two sets of clauses encoded by ZBDDs, while removing subsumed clauses.

Definition 4 (Subsumption-free union: $\sqcup$)

T_1 : $0 \sqcup A = NoSub(A)$ $\qquad$ T_2 : $1 \sqcup A = 1$

T_{3a}: $\Delta(l, A_1, A_2) \sqcup 0 = NoSub(\Delta(l, A_1, A_2))$

T_{3b}: $\Delta(l, A_1, A_2) \sqcup 1 = 1$

R_1 : *if* $l < m$, $\Delta(l, A_1, A_2) \sqcup \Delta(m, B_1, B_2) =$
$\Delta(l,\ NoSub(A_1) \setminus\!\!\setminus (A_2 \sqcup \Delta(m, B_1, B_2)),$
$A_2 \sqcup \Delta(m, B_1, B_2))$

R_2 : *if* $l > m$, $\Delta(l, A_1, A_2) \sqcup \Delta(m, B_1, B_2) =$
$\Delta(m, NoSub(B_1) \setminus\!\!\setminus (B_2 \sqcup\ \Delta(l, A_1, A_2)),$
$B_2 \sqcup \Delta(l, A_1, A_2))$

R_3 : $\Delta(l, A_1, A_2) \sqcup \Delta(l, B_1, B_2) =$
$\Delta(l,\ (A_1 \sqcup B_1) \setminus\!\!\setminus (A_2 \sqcup B_2),\ (A_2 \sqcup B_2))$

Theorem 4 *Given two sets of clauses S_1 and S_2, let us denote by $S_1 \hat{\sqcup} S_2$ the subsumption-free union of S_1 and S_2 (i.e. the result of the union of S_1 and S_2 from which all subsumed clauses have been removed). If A and B are two ZBDDs encoding sets of clauses, then we have $f_{[\![A \sqcup B]\!]} \leftrightarrow f_{[\![A]\!]} \wedge f_{[\![B]\!]}$ and $[\![A \sqcup B]\!] = [\![A]\!] \hat{\sqcup} [\![B]\!]$.*

Again, the proof is essentially based on lemma 1 and corollary 2. Note that in the case where A and B correspond to sets of clauses that are already free of subsumed clauses, then it is not necessary to apply the $NoSub$ operator in the cases T_1, T_{3a}, R_1 and R_2.

We now introduce our major operator, used for computing the ZBDD corresponding to the distribution of two sets of clauses. For a better lisibility, and since we have to pay a particular attention to the distribution of clauses containing complementary literals, we introduce the new notation $\nabla(x, A_1, A_2, A_3)$ as a shortcut of a ZBDD of the form $\Delta(x, A_1, \Delta(\neg x, B_1, B_2))$, where x corresponds to a propositional variable. Note that, because of the tautology elimination rule, A_1 cannot be labeled by $\neg x$. Thus A_1, A_2 and A_3 do only contain nodes labeled by literals of higher order than $\neg x$.

Definition 5 (Clause distribution: $\times$)

T_1 : $0 \times A = 0$

T_2 : $1 \times A = NoSub(A)$

T_{3a}: $\Delta(l, A_1, A_2) \times 0 = 0$

T_{3b}: $\Delta(l, A_1, A_2) \times 1 = NoSub(\Delta(l, A_1, A_2))$

R_1 : $\nabla(x, A_1, A_2, A_3) \times \nabla(x, B_1, B_2, B_3) =$
$\nabla(x,\ ((A_1 \times B_1) \sqcup (A_1 \times B_3) \sqcup (B_1 \times A_3)) \setminus\!\!\setminus (A_3 \times B_3),$
$((A_2 \times B_2) \sqcup (A_2 \times B_3) \sqcup (B_2 \times A_3)) \setminus\!\!\setminus (A_3 \times B_3),$
$(A_3 \times B_3))$

R_2 : *if* $label(B_2) \neq \neg x$,
$\nabla(x, A_1, A_2, A_3) \times \Delta(x, B_1, B_2) =$
$\nabla(x,\ ((A_1 \times B_1) \sqcup (A_1 \times B_2) \sqcup (B_1 \times A_3)) \setminus\!\!\setminus (A_3 \times B_2),$
$(A_2 \times B_2) \setminus\!\!\setminus (A_3 \times B_2),\ (A_3 \times B_2))$

R_3 : $\nabla(x, A_1, A_2, A_3) \times \Delta(\neg x, B_1, B_2) =$
$\nabla(x,\ (A_1 \times B_2) \setminus\!\!\setminus (A_3 \times B_2),$
$((A_2 \times B_1) \sqcup (A_2 \times B_2) \sqcup (B_1 \times A_3)) \setminus\!\!\setminus (A_3 \times B_2),$
$(A_3 \times B_2))$

R_4 : *if* $label(A_2) \neq \neg x$,
$\Delta(x, A_1, A_2) \times \Delta(\neg x, B_1, B_2) =$
$\nabla(x,\ (A_1 \times B_2) \setminus\!\!\setminus (A_2 \times B_2),$
$(A_2 \times B_1) \setminus\!\!\setminus (A_2 \times B_2),\ (A_2 \times B_2))$

R_5 : *if* $(l \in \{x, \neg x\}$, $label(A_2) \neq \neg x$, $label(B_2) \neq \neg x)$,
$\Delta(l, A_1, A_2) \times \Delta(l, B_1, B_2)=$
$\Delta(l, ((A_1 \times B_1) \sqcup (A_2 \times B_1) \sqcup (A_1 \times B_2)) \setminus\!\!\setminus (A_2 \times B_2),$
$A_2 \times B_2)$

R_6 : *if* $l < m$, $\Delta(l, A_1, A_2) \times \Delta(m, B_1, B_2)=$
$\Delta(l, (A_1 \times \Delta(m, B_1, B_2)) \setminus\!\!\setminus (A_2 \times \Delta(m, B_1, B_2)),$
$A_2 \times \Delta(m, B_1, B_2))$

R_7 : *if* $l > m$, $\Delta(l, A_1, A_2) \times \Delta(m, B_1, B_2)=$
$\Delta(m, (B_1 \times \Delta(l, A_1, A_2)) \setminus\!\!\setminus (B_2 \times \Delta(l, A_1, A_2)),$
$B_2 \times \Delta(l, A_1, A_2))$

Theorem 5 *Let A and B be two ZBDDs encoding sets of clauses, then $A \times B$ computes the ZBDD encoding the subsumption-free distribution of $[\![A]\!]$ over $[\![B]\!]$. $f_{[\![A \times B]\!]}$ is equivalent to $f_{[\![A]\!]} \vee f_{[\![B]\!]}$.*

SKETCH OF PROOF: The proof analyzes the different configurations, depending on the form of the two ZBDDs A and B. The most general case is characterized by the rule R_1, when $A = \nabla(x, A_1, A_2, A_3)$ and $B = \nabla(x, B_1, B_2, B_3)$, which respectively encode $\{x \vee [\![A_1]\!]\} \cup \{\neg x \vee [\![A_2]\!]\} \cup [\![A_3]\!]$ and $\{x \vee [\![B_1]\!]\} \cup \{\neg x \vee [\![B_2]\!]\} \cup [\![B_3]\!]$. The distribution amounts to compute recursively the distribution of each ZBDD A_1, A_2 and A_3, over each ZBDD B_1, B_2 and B_3. Let us notice yet that $A_1 \times B_2$ and $A_2 \times B_1$ occur in clauses that all contain both x and $\neg x$. Thus, they are eliminated by the tautology elimination rule. Then, among the remaining sets, we just remove clauses that may be subsumed, according to corollary 2. The remaining rules correspond to particular cases of the rule R_1.

3 Reviving the original DP procedure

The original Davis-Putnam procedure [4], proposed in 1960, is often confused with its modified DLL version of 1962 [11]. However, the two versions differ in a fundamental way. While DLL is a backtrack search procedure for SAT, DP rather amounts to a succession of cut-eliminations steps. Although generally considered as inefficient, Dechter and Rish [6] have noticed that, on some structured instances characterized by a low *induced width*, DP may perform better than DLL implementations (without backjumping and learning techniques). However, most of the time, DP produces successive *resolvents* whose size grows in a prohibitive way. Galil first proved the intractability of DP [9], using a problem for which any proof by directed resolution is exponentially long. Ten years later, Haken [10] and Urquhart [21] extended this result to all resolution based procedures. These theoretical results suggest that resolution based procedures like DP and DLL are not appropriate to solve such instances. But they rely on the implicit assumption that resolution steps are performed one by one. We here show that taking advantage of ZBDD structures for representing sets of clauses may lead to reconsider this point. In [9], Galil describes the DP procedure as:

I. Choose a propositional variable x of f.
II. Replace all the clauses of f containing x (or $\neg x$) by those which can be obtained by resolution on x.
III. a. If the empty clause is present, then the original set is unsatisfiable.
b. If the current set is empty, then the original set is satisfiable.
c. Otherwise, repeat I-III for the new set of clauses.

Let us denote by f_x^+ (resp. f_x^-) the formula corresponding to the conjunction of clauses of f that contain the literal x (resp. $\neg x$), and from which the literal x (resp. $\neg x$) has been removed. Let $f_{x'}$ denote the formula corresponding to the conjunction of clauses of f that do not contain either x or $\neg x$. Then, by factorization of x and $\neg x$, the formula f is equivalent to $(x \vee f_x^+) \wedge (\neg x \vee f_x^-) \wedge f_{x'}$, where f_x^- and $f_{x'}$ are under cnf. The second step of DP then amounts to replace $(x \vee f_x^+) \wedge (\neg x \vee f_x^-) \wedge f_{x'}$ by the cnf $(f_x^+ \vee f_x^-) \wedge f_{x'}$. This is the real source of complexity since distributing f_x^+ over f_x^- can generate $|f_x^+|.|f_x^-|$ clauses. However, in terms of ZBDD, this may be performed efficiently using the $\times$ operator. If we denote by $\zeta(f)$ the ZBDD encoding the set of clauses of f, the result of the cut-elimination of a variable x can then be obtained with our operators, by computing:

$$\textit{cut-elimination}(f, x) \;=\; (\,\zeta(f_x^+) \;\times\; \zeta(f_x^-)\,) \;\sqcup\; \zeta(f_{x'})$$

The main advantage is that the cost of this operation does not depend on the number of clauses, but only on the number of nodes of $\zeta(f_x^+)$, $\zeta(f_x^-)$ and $\zeta(f_{x'})$, which can be constructed at low cost from $\zeta(f)$, using standard ZBDD operations. As a consequence, it is possible to perform in a single step many resolutions involving a huge number of clauses, if the corresponding ZBDDs have reasonable sizes (we call this multi-resolution). The use of such data structures together with the $\times$ operator leads to completely reconsider the general consensus on DP inefficiency. Another point worth reconsidering is the heuristics used for choosing the cut variable, in step I. With more conventional data structures, a traditional criterion is to select a variable which cut-elimination produces the minimal number of new clauses [3]. Of course, since the number of nodes of the ZBDD is bounded by the size of the formula, this remains an interesting idea (if not specified, this is our default heuristic). But since the size of ZBDDs is not necessarily directly related to the number of clauses, new criteria may be also be investigated.

This revived DP procedure has been implemented to evaluate the effectiveness of this approach. The resulting system, called ZRES, uses the Cudd package [19], which includes basic ZBDD operations, as well as useful dynamic

Table 1. Cpu time on Hole-n instances

Instances	ASAT	SATO	ZRES *std*	ZRES *cd*
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

reordering functions. Two versions have been realized. The standard version (*std*) uses the standard ZBDD distribution operators and deletes tautologies and subsumed clauses afterwards. The other version (*cd*, for *clause-distribution*) uses our $\times$ operator to perform the three steps simultaneously.

4 Solving two hard problems for resolution

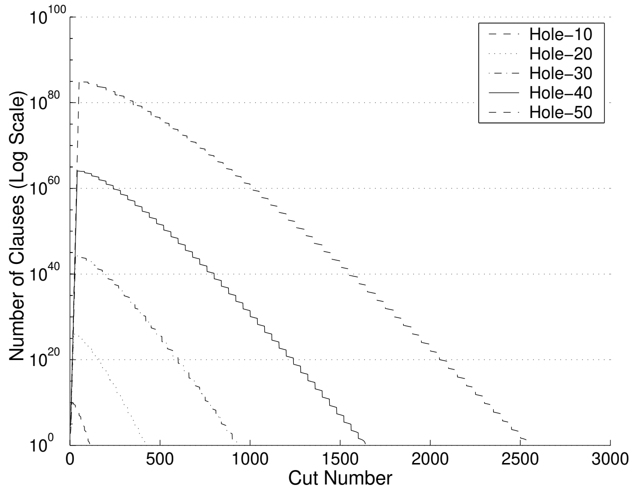
This section presents empirical and theoretical results obtained with ZRES on two hard problems for resolution: Pigeon Hole [10] and Urquhart [21]. Since for these problems DP must generate an exponential number of clauses, they seem good candidates to evaluate the ability of our data structures to compress large sets of clauses. All tests have been performed on a Linux Pentium-II 400MHz[3] with 256MB. Our results are compared, when possible, with those of two DLL implementations: ASAT [8], a good-and-simple DLL implementation, and SATO 3.0 [22], which includes many optimizations such as backjumping and conflict memorization, and which, to the best of our knowledge, has obtained the best results on Dimacs benchmarks [7, 18]. All values correspond to cpu times and are expressed in seconds. We assume that an instance that cannot be solved in less than 10000 seconds counts for 10000s.

4.1 The Pigeon Hole problem

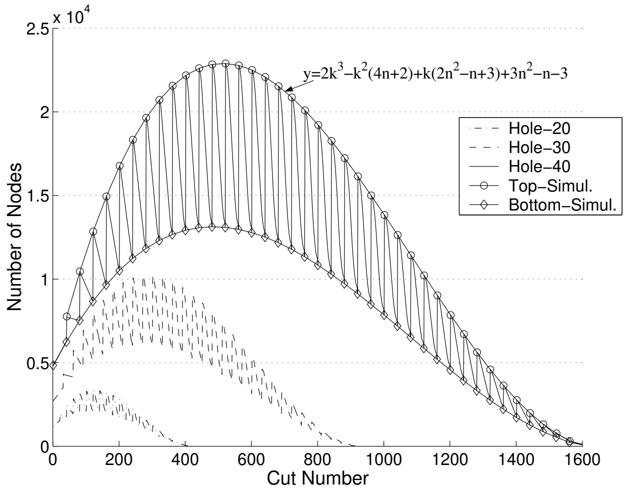
The well-known Pigeon Hole problem has received much attention since Haken [10] used it to prove the intractability of resolution. The simplicity of this problem, which states that $n+1$ pigeons cannot fit in n holes, contrasts with its intractability. Results of our experiments are summarized table 1. Surprisingly, the best results for this problem have been obtained with a heuristic function which tends to maximize the number of clauses produced at each step. This confirms that usual DP heuristics may not remain appropriate with ZBDDs structures (ZRES cannot solve this problem with the default heuristics).

The first observation is that ZRES *cd* not only surpasses ASAT and SATO, but also solves much bigger instances,

[3]On the Dimacs [7] machine scale benchmark, this machine has a user time saving of 305%.



a. Number of clauses



b. Number of nodes

Figure 3. Pigeon Hole resolvents

that cannot be solved by the DLL procedures. ZRES even managed to solve Hole-55 (84756 clauses) in less than 2 days. In comparison, Hole-10 only contains 561 clauses[4]. The second observation concerns the *std* and *cd* versions of ZRES. Results confirm the effectiveness of our specialized $\times$ operator. The savings obtained with the second version even seem to increase as the size of the problem augments.

One may notice, on figure 3-a, that, from the clause point of view, the process is composed of two phases. The first phase corresponds to the first n cuts and leads to a super exponential growth. By studying precisely which cuts are performed, it can be shown that, at the end of the first phase, the total number of clauses is n^n. The second phase corresponds to a regular step by step decreasing (each of the

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

$n+1$ steps is n cuts long). To understand how ZRES manages with such a huge number of clauses, we compare the respective variations of the total number of clauses (figure 3-a) with respect to the total number of nodes (figure 3-b). At the end of the first phase, the latter has only increased by 10%. During the second phase, the number of nodes looks like oscillating between two other curves, corresponding to the minimal and maximal values for each period (a period corresponds to n cuts). During the first cut of each period, the number of nodes jumps from its minimal to its maximal value. During the $n-1$ remaining cuts of the period, it decreases regularly until the minimal value of the next period. These oscillations clearly suggest the repetition of the same phenomenon, closely related to the structure of the formula. Indeed, sudden increases of the number of nodes correspond to eliminations of variables that break some symmetries and reduce the number of shared nodes, while the remaining cuts of a period progressively restore the regularities.

The shape of the top envelope also contrasts with the variation of the total number of clauses. It increases progressively and reaches its maximum after approximatively $\frac{n^2}{3}$ cuts. We have conducted an analytical study of the structure of successive ZBDDs, to evaluate the successive maximal and minimal values. The results confirm those obtained by regression analysis on experimental values and prove that the max and min envelopes correspond to polynomial functions of degree 3. On figure 3-b, we represent, for the Hole-40 case, the curve corresponding to the top and bottom envelopes and give the precise equation characterizing the top envelope obtained by analytical study, which should be read with $k = \lfloor \frac{x}{n} \rfloor$.

Theorem 6 *For the Pigeon Hole problem, there exists a literal ordering and a cut-elimination ordering that lead to an exponential explosion of the number of clauses, while the number of nodes in the ZBDD encoding remains bounded by a polynomial function of degree* 3.

The proof amounts to characterizing the structure of the set of clauses and of the corresponding ZBDD at the end of the first phase, at the beginning of each period, and after the k^{th} cut $(1 < k < n)$ of each period.

Despite bad complexity results characterizing DP for the Pigeon Hole problem, ZRES seems effective for solving this problem.

4.2 The Urquhart problem

The Urquhart problem [21] encodes a property of expander graphs, in a modified version of those initially proposed by Tseitin [20]. Galil [9] also used such graphs to exhibit an exponential lower bound on the cost of satifiability testing using directed resolution. Practically, Urquhart's instances correspond to a set of connected sub-bases. Each sub-base is the cnf translation of some *biconditional formula* (a chain of equivalences of the form $l_1 \leftrightarrow l_2 \leftrightarrow \ldots \leftrightarrow l_k$). As a consequence, all clauses of a given sub-base have the same length, are built using the same variables (occurring positively or negatively) and their number of negated literals are either all odd or all even. Again, we may observe on table 2 that ZRES performs very well on such examples.

Table 2. Cpu time on 100 Urq-3 instances

System	Total	# Solved	Mean (solved)
ASAT	404 287	69	1366
SATO	776 364	26	1398
ZRES *cd*	69.2	100	0.69

Table 3. Mean on 1000 Urquhart instances.

Urq	Initial Var Nb	Initial Cl Nb	Max Cl Nb	Max Nodes Nb	ZRES Time
4	78	714	10^{10}	440	1.72
6	178	1683	10^{24}	1011	8.88
8	318	3024	10^{43}	1808	29.6
10	498	4754	10^{67}	2833	72.0

We also tested bigger instances with the default heuristics. Results (table 3) clearly show that the maximum number of clauses handled in each calculus is exponentially higher than the maximum number of nodes reached. Note that here, we do not report the time of ZRES *std* time, because for this class of problem, no significant difference may be observed.

Because the Urquhart problem rather corresponds to a general framework, a precise analytical study is not possible. But it is still possible to have some intuition on the reason why ZRES still cope with such instances. By construction, each sub-base built on i variables contains 2^{i-1} clauses. But this highly structured set of clauses can be encoded by a ZBDD having only $O(i)$ nodes. Successive cuts amplify this exponential difference. As a matter of fact, a distribution between the clauses of two different sub-bases, built respectively on i and j variables, leads to 2^{i+j-2} clauses, which can be encoded with only $O(i+j)$ nodes. We thus have the following theorem:

Theorem 7 *Biconditional formulæ on* n *variables have a cnf representation of exponential size* $(n.2^{n-1})$ *but can be encoded by a ZBDD of exactly* $2.n-2$ *nodes.*

Again, thanks to the high compression capabilities of ZBDDs, this second historical problem, in proving intractability of resolution, seems practically solvable using our DP implementation.

Table 4. 3-SAT, 190 Cl, 44 Var, 1000 exp.

ZRES	cpu time	Max. Node Num.	Memory (MB)
std	306	5 638 848	186
cd	118	284 474	13

5 Other experimental results

5.1 ZRES *std* vs ZRES *cd*

To evaluate the benefit of using the $\times$ operator, we have performed an experimental comparison between ZRES *std* and ZRES *cd* on random sets of clauses. We use the fixed clause-length model [14], which is a reference problem in the literature. Two conclusions may be drawn from this experiment[5]. First, the obtained results (see table 4) confirm that ZRES is definitely not a good candidate for solving such unstructured instances. The work of [3] already pointed out that DP is not adequate for random instances. In this context, improvements due to ZBDDs are poor. Because they lack regularities, random instances are not really compressible. Nevertheless, we still may observe a significant difference between the two versions of ZRES. The explanation may be found in [3] which pointed out the importance of subsumption checking on such examples for DP. Actually, the use of the specialized $\times$ operator, instead of usual ZBDD operators, leads to significant savings, especially in memory.

5.2 Compression capabilities of ZBDDs

Using ZBDDs for representing sets of clauses is motivated by the hypothesis that structured instances should present some regularities, that should also be observable when expressed under cnf. A possible way to validate such a hypothesis is to consider the *compression rate* induced by this representation, defined as the ratio of the total number of nodes of the ZBDD encoding the cnf, with the total number of literals of the cnf. We measured the variation of this compression rate during the resolution of random generated as well as structured instances.

Experiments on 1000 3-SAT instances with 42 variables and 180 clauses show that the initial compression rate is almost always the same: 2.05. During the successive cuts, it increases up to 6.15 and then decreases again. It reaches its maximal value when handling the maximal number of clauses. It might be surprising to obtain compression rates greater than 1 on random instances. But, given the characteristics of the generated instances, there is some statistical evidence that clauses begin and/or end in the same way. During the calculus, the rate increase may be explained by the duplication of portions of clauses during the successive cuts. Thus, generated resolvents do no more have the caracteristics of the initial random instances.

[5] ASAT and SATO results do not appear in the table, because both solve such instances instantaneously

Table 5. Compression rates (std. dev.)

Problems	Beginning	Best
Alea-130c-30v	2.06 (0.02)	4.58 (0.34)
Alea-180c-42v	2.05 (0.02)	6.15 (0.44)
Pret150	1.88 (0)	1157 (0)
Par32-c	2.14 (0.02)	10^7 (10^7)
Ii8	2.43 (1.09)	10^7 (10^7)
Flat-50	1.76 (0.01)	9.61 (2.14)
Ssa	1.72 (0.01)	5.72 (0.28)
Massacci-R4	7.16 (0.25)	72.9 (8.95)
Aralia	1.79 (0.28)	10^6 (10^7)

For structured instances, results often differ in a significant way. Most of the time, the compression rate has a low initial value but reaches much higher values. This is the case for $ii8$, $pret150$ and *par32-c* from Dimacs benchmarks as well as on the problem of Massacci [12], concerning a cryptography application. The ssa and $flat$ classes are however exceptions. But, when looking closely at the characteristics of these classes of problems, it is interesting to note that even if they correspond to real problems (graph coloring and circuit testing), both kinds of instances seem to incorporate a random factor in their generation process. This could be a possible explanation for such differences. We also used the Aralia benchmark [15], which is seldom used to evaluate SAT solvers. For these instances, the goal to achieve is only to eliminate a subset of the variables of the initial formula and to produce the resolvent. This task perfectly suits to ZRES. This benchmark is interesting because its instances correspond to models of physical systems. Again, the compression rates are quite different from those of random instances.

If these results confirm that ZBDDs are effective for compressing large structured sets of clauses, a high compression rate doesn't mean that ZRES is able to solve the problem efficiently. In particular, ZRES cannot achieve the resolution of *Massaci-R4*, $ii8$ and of *Parity-32*. More generally, for the Dimacs benchmarks, SATO remains quicker.

6 Conclusion

In this paper, we have proposed using ZBDDs to encode sets of clauses efficiently. We have shown that memory savings due to this encoding may become very high when considering structured instances of clauses. We have introduced specialized operators for clause set handling and shown that the $\times$ operator may distribute sets of clause of exponential size, while handling polynomial size data structures. This makes it possible to reconsider the original DP procedure and to propose a revised version, which is able to perform multi-resolution on large sets of clauses, in an efficient manner. This allows this new DP to solve hard instances for resolution, which contrasts with the widespread idea concerning its inefficiency.

Exponential lower bounds on the length of proofs have often led to consider resolution based procedures as intrinsically limited. It is often suggested that the way to surpass such limitations, which occur even on simple problems like Pigeon Hole, is to switch to more powerful proof systems. Our feeling is that, in some cases, the length of a proof is not a good indicator of its complexity, as only the length of a message is not significant in information theory.

Many directions are interesting for further work. We have used our $\times$ operator at the heart of a SAT prover. The introduction of ZBDDs give the opportunity to consider the DP algorithm from a different point of view. New heuristics, based on the ZBDD structure, instead of the number of clauses, have to be developed. We think that this $\times$ operator can also be used to solve efficiently other problems such as computing sets of prime implicates. Preliminary work on this point seems promising. From the SAT viewpoint, we think that the DLL procedure could also take advantage of ZBDD representation to handle very large sets of clauses in a compact way. A further perspective could be to develop a hybrid algorithm, merging DP and DLL, to benefit from the best of both approaches.

References

[1] O. Coudert J.-C. Madre. A new method to compute prime and essential prime implicants of boolean functions". In T. Knight J. Savage, editor, *Advanced Research in VLSI and Parallel Systems*, pages 113–128, March 1992.

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

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

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

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

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

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

[8] Olivier Dubois. Can a very simple algorithm be efficient for SAT? Electronically available on [7].

[9] Zvi Galil. On the complexity of regular resolution and the Davis-Putnam procedure. *Theorical Computer Science*, 4:23–46, 1977.

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

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

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

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

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

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

[16] A. Rauzy. Mathematical foundations of minimal cutsets. *IEEE Transaction on Reliability*. (to appear).

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

[18] Sat-Ex site. http://www.lri.fr/˜simon/satex/satex.php3.

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

[20] G. Tseitin. On the complexity of derivation in propositional calculus. *studies in Constructive Mathematics and Mathematical Logic, part 2*, pages 115–125, 1968.

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

[22] Hantao Zhang. SATO: An efficient propositional prover. In *CADE-14*, LNCS 1249, pages 272–275, 1997.