Empirical evaluation of two versions of the Davis-Putnam algorithm

Rina Dechter  
Information and Computer Science  
University of California, Irvine  
dechter@ics.uci.edu

Irina Rish  
Information and Computer Science  
University of California, Irvine  
irina@ics.uci.edu

Abstract

This paper summarizes the author’s paper (presented in KR-94 and titled “Directional Resolution: The Davis-Putnam Procedure, Revisited” [Dechter and Rish, 1994]), focusing on its empirical aspects. It compares a resolution based variant of the Davis-Putnam algorithm with a backtracking variant and highlights their sensitivity (or insensitivity) to the problems’ structure. Experiments show that the behaviour of the algorithms on problems with special topologies, like chains, differs dramatically from that on random k-cnf problems.

Introduction

In 1960, Davis and Putnam [Davis and Putnam, 1960] presented their resolution algorithm. Two years later, Davis, Logemann, and Loveland proposed a modification of that algorithm: the elimination rule (rule III in [Davis and Putnam, 1960]) in the original Davis-Putnam algorithm was replaced by the splitting rule (rule III' in [Davis et al., 1962]) in order to avoid the memory explosion encountered when testing the original algorithm empirically. However, this seemingly minor syntactic change completely changes the nature of the algorithm. The original algorithm (called here DP-elimination) performs resolution along some order of the atomic formulas, and thus belongs to a class of elimination algorithms. Its competitor (called DP-backtracking), searches through the space of possible truth assignments and belongs to the class of backtracking search algorithms (e.g., dynamic backtracking). It implements a form of dynamic variable ordering by performing unit resolution until quiescence at each step. Analyses of the original Davis-Putnam algorithm have emphasized its worst-case exponential behavior [Galil, 1977, Goerdt, 1992], while neglecting its virtues. Consequently, it was overshadowed by its competitor, DP-backtracking, and most work on the Davis-Putnam procedure, from then on, quotes the backtracking version [Goldberg et al., 1982, Selman, 1992], wrongly suggesting that this is the algorithm presented in [Davis and Putnam, 1960].

In [Dechter and Rish, 1994] we present a variant of DP-elimination called directional resolution, provide theoretical analysis and empirical evaluation of that algorithm. We show that, unlike DP-backtracking, DP-elimination’s performance can be bounded as a function of w*, a parameter measuring the sparseness of its interaction graph. This bound leads to the identification of a class of theories, having a chain-like structure, for which directional resolution is an effective procedure, and for which DP-backtracking is ineffective.

The main conclusion we draw is that any general algorithm for satisfiability should include a structure exploiting component, or else it is bound to fail on some, relatively easy problem instances. Directional resolution is just one algorithm that exploit the structure, not necessarily the best one (for satisfiability). It should be noted that directional resolution’s worst-case and average case complexity are quite close. However, backtracking algorithms when augmented with similar structure exploiting features (like backjumping + learning) may yield the same worst-case guarantees with a much better average case performance [Frost, 1994].

Our second claim is that, unlike previous analyses, directional resolution is effective for sparse theories. It should not, however, be judged as a satisfiability algorithms but rather as a knowledge compilation procedure. The algorithm solves a harder problem than satisfiability, it generates an equivalent theory that is backtrack-free, and thus is effective for model checking and query processing.

1 Two Versions of the Davis-Putnam Algorithm

Both versions of the Davis-Putnam algorithm take as an input a propositional theory \( \varphi \) in conjunctive nor-
directional-resolution

**Input:** A cnf theory \( \varphi \), an ordering \( d = Q_1, \ldots, Q_n \) of its variables.

**Output:** A decision of whether \( \varphi \) is satisfiable. If it is, a theory \( E_d(\varphi) \) equivalent to \( \varphi \), else an empty directional extension

1. Initialize: generate an ordered partition of the clauses, \( \text{bucket}_1, \ldots, \text{bucket}_n \), where \( \text{bucket}_i \) contains all the clauses whose highest literal is \( Q_i \).
2. For \( i = n \) to 1 do:
3. Resolve each pair \( \{ (\alpha \lor Q_i), (\beta \lor -Q_i) \} \subseteq \text{bucket}_i \). If \( \gamma = \alpha \lor \beta \) is empty, return \( E_d(\varphi) = \emptyset \), the theory is not satisfiable; else, determine the index of \( \gamma \) and add it to the appropriate bucket.
4. End-for.
5. Return \( E_d(\varphi) = \bigcup \text{bucket}_i \).

Figure 1: Algorithm directional resolution

We use the following notation: propositional symbols, also called variables, will be denoted by uppercase letters \( P, Q, R, \ldots \), and disjunctions of literals, or clauses, by \( \alpha, \beta, \ldots \). The clause whose literal appears in either \( \alpha \) or \( \beta \) will be denoted as \( (\alpha \lor \beta) \). The resolution operation over two clauses \( \{ (\alpha \lor Q) \) and \( (\beta \lor -Q) \) results in a clause \( (\alpha \lor \beta) \), thus eliminating \( Q \).

1.1 DP-Elimination – Directional Resolution

Algorithm directional resolution (DR) (the core of DP-elimination [Davis and Putnam, 1960]) is described in Figure 1. We call its output theory, \( E_d(\varphi) \), the directional extension of \( \varphi \). The algorithm can be conveniently described using a partitioning of the set of clauses of a theory into buckets. Given an ordering \( d = Q_1, \ldots, Q_n \), all the clauses containing \( Q_i \) that do not contain any symbol higher in the ordering will be put in the bucket for \( Q_i \) (bucket\(_i\)). Given the theory \( \varphi \), algorithm DR processes the buckets in a reverse order of \( d \). When processing bucket\(_i\), DR resolves over \( Q_i \) all possible pairs of clauses in the bucket and inserts the resolvents into the appropriate lower buckets. According to [Davis and Putnam, 1960], a theory has a non-empty directional extension iff it is satisfiable.

It was shown that, if the extension \( E_d(\varphi) \) is not empty, any model of \( \varphi \) can be generated in time \( O(|E_d(\varphi)|) \) in a backtrack-free manner, consulting \( E_d(\varphi) \), as follows:

Step 1. Assign to \( Q_1 \) a truth value that is consistent with clauses in \( \text{bucket}_1 \), if the bucket is empty, assign \( Q_1 \) an arbitrary value; Step i. After assigning a value to \( Q_1, \ldots, Q_{i-1} \), assign to \( Q_i \) a value that, together with the previous assignments, will satisfy all the clauses in \( \text{bucket}_i \).

It was shown [Dechter and Rish, 1994] that the time complexity of directional resolution is \( O(n \cdot |E_d(\varphi)|^2) \), where \( n \) is the number of propositional letters in the language, \( E_d(\varphi) \) is the output theory (extension) of \( \varphi \). The size of the output theory can be bounded using a topological parameter called width defined relatively to the interaction graph of the theory. The interaction graph of \( \varphi \) is an undirected graph that contains one node for each propositional variable and an arc connecting any two nodes whose associated variables appear in the same clause. Given an ordering of variables (nodes), the width of a node \( x \) is the number of its parents, i.e., nodes connected to \( x \) and preceding \( x \) in the ordering, the width of an ordering is the maximum width of all nodes, and the width of a graph is the minimum width of all orderings of that graph. Given a graph \( G \) and an ordering \( d \), the graph generated by recursively connecting the parents of nodes in \( G \), in reverse order of \( d \), is called induced graph of \( G \) w.r.t. \( d \). The width of induced graph is called the induced width of \( G \) w.r.t. \( d \) and is denoted by \( w^*(d) \).

The size of directional extension and, therefore, its time complexity is exponential in \( w^* \) [Dechter and Rish, 1994].

**Theorem 1:** Let \( \varphi = \varphi(Q_1, \ldots, Q_n) \) be a cnf, \( G(\varphi) \) its interaction graph, and \( w^*(d) \) its induced width along \( d \); then, the size of \( E_d(\varphi) \) is \( O(n \cdot 3^{w^*(d)}) \).

Therefore, directional resolution is efficient for theories with bounded induced width. In the next section we introduce a class of such theories, namely chains.

The notion of induced width leads to several heuristic orderings. One such ordering is min-width: given a graph, select a variable with the smallest degree, put it last in the ordering, eliminate that node from the graph and then continue with the remaining graph.

Another heuristic ordering is min-diversity. Given a theory \( \varphi \) and an ordering \( d \), let \( Q_i^+ \) (or \( Q_i^- \)) denote the number of times \( Q_i \) appears positively (or negatively) in bucket\(_i\), relative to \( d \). The diversity of \( Q_i \) relative to \( d \) is \( Q_i^+ \times Q_i^- \). Diversity of \( Q_i \) estimates the number of possible resolutions performed in the \( i \)-th bucket.

Min-diversity ordering is constructed from last to first by choosing a variable with the smallest diversity on each step.

1.2 DP-backtracking

DP-backtracking algorithm we are referring to in this paper is a variant of the Davis-Putnam procedure (see Figure 2) augmented with the 2-literal clause heuristic proposed in [Crawford and Auton, 1993]. This heuristic prefers a variable that would cause the largest number of unit propagations. The number of possible unit propagations is approximated by the number of 2-literal clauses in which the variables appear. The modified version significantly outper-
forms DP-backtracking without this heuristic (see also
[Crawford and Auton, 1993]).

In order to find a solution following DR we ran DP-
backtracking using the reverse ordering of variables
used by DR, but without the 2-literal clause heuristic.
The reason is that we wanted to fix the order of
variables. As theory dictates, no deadends occur when
DP-backtracking is applied after DR on the same
ordering. In this case DP-backtracking takes linear
time in the extension size.

2 Experimental evaluation

We tested the algorithms on uniform k-cnfs and on
chain-like cnfs. To generate uniform k-cnfs we used
the generator proposed by [Mitchell et al., 1992] tak-
ing as input the number of variables n, the number
of clauses m, and the number of literals per clause k.
We generate each clause randomly choosing k variables
from the set of n variables and by determining the po-
ularity of each literal with probability 0.5. We also gen-
erated mixed theories containing clauses of length k_1
and clauses of length k_2. Our second generator, called
chains, first used the uniform k-cnf generator to ob-
tain a sequence of n independent random subtheories,
and then connected all the subtheories in a chain by
generating 2-cnf clauses using one variable from the
i-th subtheory and one from the (i + 1)-th subtheory.
Similarly we also connected the n independent subthe-
ories into a tree structure. The obtained results were
similar to those on chains, so we report only the result
on chains.

The algorithms were tested using the following order-
ings: input ordering as used by the problem generator,
min-width ordering and min-diversity ordering.

We recorded the CPU time for all algorithms, the num-
ber of deadends for DP-backtracking, the number of
new clauses generated by DR, the maximal size of
generated clauses, and the induced width. The number of
experiments shown in the figures is per each point.

2.1 Results for uniform k-cnfs

We compared DP-backtracking with DR on randomly
generated k-cnfs for k=3,4,5 and on mixed theories
for finding one solution only. In all these cases DP-
backtracking significantly outperforms DR. It is ob-
erved that the complexity of DR indeed grows expo-
entially with the size of problems (see Figure 3(a)).
We show the results for 3-cnfs with 20 variables only.
On larger problems DR often ran out of memory be-
cause of the large number of generated clauses.

Since DR was so inefficient for solving uniform k-cnfs
we also experimented with Bounded Directional Res-
olution (BDR) using different bounds. Given a bound
k, the algorithm records clauses of size k or less. Con-
sequently, its complexity is polynomial in k. Our ex-
periments show that when the input theory is a uni-
form k-cnf and BDR uses a bound less than k, almost
no new clauses are added (this may not be so obvious
a priori, because different clauses could have common
literals). On the other hand, when the bound is strictly
greater than k, the preprocessing phase of BDR by it-
self is considerably worse than DP-backtracking. The
only promising case (for satisfiability) occurs when the
bound equals k. We observed that in this case rela-
tively few clauses were added by BDR which there-
fore ran much faster. Also, DP-backtracking was of-
ten faster on the generated theory and therefore the
combined algorithm was slightly more efficient than
DP-backtracking alone (see Figure 3(b)).

2.2 Results for chains

The behaviour of the algorithms on chains differs dra-
matically from that on uniform instances. We found
extremely hard instances for DP-backtracking, orders
of magnitude harder than those generated by the un-
iform model. In Table 1 we contrast the performance
of DP-backtracking on uniform 3-cnf problems and on
3-cnf chain problems of the same size. The chain pro-
blems have 25 subtheories with 5 variables and 9-23
clauses per subtheory, together with 24 2-cnf clauses
connecting the subtheories in the chain. The corre-
sponding uniform 3-cnf problems have 125 variables
and 249 to 599 clauses. We tested DP-backtracking
on both classes of problems. The table presents the
mean values on 20 experiments per problem size. We
used min-diversity ordering for each instance.

As observed, the number of deadends across the cross-
over point for chains is orders of magnitude higher than
for uniform 3-cnfs.

In contrast, DR was fast on chains, sometimes more
than 1000 times faster than DP-backtracking. Table 2
compares DP-backtracking with DR on the same chain
problems as in Table 1 for the task of finding one solu-
tion and for deciding satisfiability only. Note that the
induced width is bounded. Table 3 lists the results on
Table 1: DP on uniform 3-cnfs and on chain problems of the same size: 125 variables

<table>
<thead>
<tr>
<th>Num of clauses</th>
<th>Uniform 3-cnfs</th>
<th>3-cnf chains</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>% Sat</td>
<td>Time 1st solution</td>
</tr>
<tr>
<td>249</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>349</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>399</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>449</td>
<td>100</td>
<td>0.4</td>
</tr>
<tr>
<td>499</td>
<td>95</td>
<td>3.7</td>
</tr>
<tr>
<td>549</td>
<td>35</td>
<td>8.5</td>
</tr>
<tr>
<td>599</td>
<td>0</td>
<td>6.6</td>
</tr>
</tbody>
</table>

selected hard instances from Table 2 (number of deadends exceeds 4000). Note that almost all the hard chain problems for DP-backtracking were unsatisfiable.

Table 3: DR and DP on hard instances (number of deadends > 4000): 3-cnf chains with 125 variables

<table>
<thead>
<tr>
<th>Num of cns</th>
<th>SAT: 0 or 1</th>
<th>DP-backtracking</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time: 1st solution</td>
<td>Dead ends</td>
</tr>
<tr>
<td>349</td>
<td>0</td>
<td>41163.8</td>
<td>3739913</td>
</tr>
<tr>
<td>349</td>
<td>0</td>
<td>1002615.3</td>
<td>9285160</td>
</tr>
<tr>
<td>349</td>
<td>0</td>
<td>550585.5</td>
<td>5105441</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>74.8</td>
<td>6053</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>87.7</td>
<td>7433</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>149.3</td>
<td>12301</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>37903.3</td>
<td>3079997</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>11877.6</td>
<td>975470</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>52.0</td>
<td>4215</td>
</tr>
<tr>
<td>399</td>
<td>0</td>
<td>814.8</td>
<td>70057</td>
</tr>
<tr>
<td>449</td>
<td>1</td>
<td>655.5</td>
<td>47113</td>
</tr>
<tr>
<td>449</td>
<td>0</td>
<td>60.5</td>
<td>4359</td>
</tr>
<tr>
<td>449</td>
<td>0</td>
<td>2549.2</td>
<td>184504</td>
</tr>
<tr>
<td>449</td>
<td>0</td>
<td>289.7</td>
<td>21246</td>
</tr>
</tbody>
</table>

We also experimented with the actual code of tableau [Crawford and Auton, 1993]. Crawford and Auton’s implementation of Davis-Putnam procedure with various heuristics. The behaviour on chains was similar to our implementation of DP-backtracking.

This may be explained as follows: suppose there is an unsatisfiable subtheory $U$ in a chain problem whose variables are put at the end of an ordering. If all the subtheories whose variables appear earlier in the ordering are satisfiable, then DP-backtracking will try...
Table 2: DR and DP on 3-cnf chains: 25 subtheories, 5 variables in each (total 125 variables)

<table>
<thead>
<tr>
<th>Num of</th>
<th>SAT:</th>
<th>DP-backtracking</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>%</td>
<td>Time: 1st solution</td>
<td>Dead ends</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.3</td>
<td>0</td>
</tr>
<tr>
<td>249</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>249</td>
<td>100</td>
<td>0.4</td>
<td>1</td>
</tr>
<tr>
<td>349</td>
<td>70</td>
<td>9945.7</td>
<td>908861</td>
</tr>
<tr>
<td>349</td>
<td>25</td>
<td>2551.1</td>
<td>207896</td>
</tr>
<tr>
<td>459</td>
<td>15</td>
<td>185.2</td>
<td>13248</td>
</tr>
<tr>
<td>499</td>
<td>0</td>
<td>2.4</td>
<td>160</td>
</tr>
<tr>
<td>549</td>
<td>0</td>
<td>0.9</td>
<td>9</td>
</tr>
<tr>
<td>599</td>
<td>0</td>
<td>0.1</td>
<td>6</td>
</tr>
</tbody>
</table>

To re-instantiate variables from the satisfiable subtheories each time it encounters a deadend caused by the unsatisfiable subtheory, DR, on the other hand, easily recognizes the unsatisfiable subtheory by performing a bounded number of resolutions when the induced width (determined by the size of subtheories) is bounded. Not knowing the structure hurts DP-backtracking. Choosing the right ordering would help but this may be hard to recognize without some preprocessing, which is exactly what DR is doing. Other variants of backtracking that are capable of exploiting the structure like backjumping [Gashnik, 1979], [Dechter, 1990], [Prosser, 1993] would avoid useless re-instantiation. Experiments with backjumping chains showed indeed that all the problems that were hard for DP-backtracking were quite easy for backjumping (see Figure 3(c)). Backjumping also outperforms DR. On the other hand, DP-backtracking was better than both backjumping and DR for easy under- and over-constrained instances.

3 Related work and conclusions

Our empirical tests show that, while on uniform theories directional resolution is ineffective, on problems with special structures, like chains, having low width, directional resolution greatly outperforms DP-backtracking which is one of the most effective satisfiability algorithm known to date. We have also experimented with other variants of backtracking like backjumping [Gashnik, 1979], [Dechter, 1990], [Prosser, 1993] that are capable of exploiting the structure. Our experiments show that both backjumping and directional resolution significantly outperform DP-backtracking on chains.

Nevertheless, directional resolution is very inefficient on uniform theories, and cannot be advocated as an effective method for general satisfiability problems. What we do advocate is that structure-based components should be integrated, together with other heuristics (like unit propagation), into any algorithm that tries to solve satisfiability effectively.

Acknowledgements

We would like to thank Dan Frost for experimenting with backjumping and Eddie Schwab for commenting on this paper.

References


