To Guess or to Think? Hybrid Algorithms for SAT

Irina Rish and **Rina Dechter**
Information and Computer Science
University of California, Irvine
{*dechter,irinar*}*@ics.uci.edu*
http://www.ics.uci.edu/{ *~irinar, ~dechter*}
phone: (714)824-6556, fax: (714)824-4056

Abstract

Complete algorithms for solving propositional satisfiability fall into two main classes: backtracking (e.g., the Davis-Putnam Procedure [4]) and resolution (e.g., Directional Resolution [9]). Roughly speaking, backtracking amounts to "guessing" (making assumption), while resolution invokes "thinking" (inference). Experimental results show that both "pure guessing" and "pure thinking" might be inefficient. We propose an approach that combines resolution and backtracking and yields a family of hybrid algorithms, parameterized by a bound on the "effective" amount of resolution allowed. The idea is to divide the set of propositional variables into two classes: *conditioning* variables, which are assigned truth values, and *resolution* variables, which are resolved upon. We report on preliminary experimental results demonstrating that on certain classes of problems hybrid algorithms are more effective than either the Davis-Putnam Procedure or Directional Resolution in isolation.

Keywords: propositional satisfiability, computational complexity, resolution, backtracking, graph representation of a propositional theory, induced width.

1 Introduction

One of the most common approaches for solving constraint satisfaction problems (CSPs) is combining search with local consistency enforcing. Backtracking search may be viewed as a systematic "guessing", as opposed to consistency enforcing which may be viewed as "thinking" as it involves making inferences. A key to having efficient constraint satisfaction algorithms lies in finding a good balance between these two activities.

The problem of propositional satisfiability is a special case of CSP, where the domains are restricted to $\{true, false\}$. The well-known Davis-Putnam Procedure (DP)[4] is a backtracking algorithm enhanced by unit resolution performed at each level of the search, which is a form of forward checking (roughly equivalent to arc-consistency). Directional Resolution (DR)[9], a variation on the original Davis-Putnam Algorithm[5], is a variable-elimination algorithm (like adaptive-consistency for CSPs[8]) applied to propositional satisfiability. The time and space complexity of DR is $O(exp(w^*))$, where w^* is a parameter of the interaction graph of the theory, called induced-width or tree-width [9]. It was not surprising, therefore, that DR was found to be significantly less efficient than DP when applied to uniformly generated 3-cnfs having large w^*, while outperforming DP, sometimes by many orders of magnitude, when applied to structured theories with bounded w^*. This last phenomenon occurred on structured theories when DP encountered rare, but exceptionally hard problems [14]. Since those problems had small w^* they were all easy for DR. This complementary behavior of DP and DR prompted the idea of combining the advantages of both algorithms. Indeed, we have shown that preprocessing DP by a bounded version of DR results in an algorithm *BDR-DP* [9] which is superior to either of its components. In this paper we pursue the idea of hybrid algorithms further.

We wish to come up with an algorithm that adjusts automatically to the problem instance. For instances likely to be solved efficiently by DP it should reduce to DP, while for favorite instances of DR it will reduce to DR. Otherwise, for instances which are not likely to be solved effectively by either DP nor DR, the algorithm will exploit a middle ground between these two extremes yielding a better algorithm than both.

To that aim, we present a family of parameterized hybrid algorithms using a parameter that controls the levels of DP and DR in the combination, and study their properties. Our preliminary results suggest that the selection of the right hybrid algorithm may be guided by the induced width of the input theories and that a selected number of them may be adjustable to the problem instance yielding a better algorithm overall. Our combined algorithms have the properties that, like DR, they produce an output theory from which a portion of the set of models can be extracted in linear output time, while, like DP, they are more space-efficient than DR [7].

In general, conditioning facilitates integration. It is a universal principle that is used to reduce a general problem into a tractable class. Here we focus on tractable problems whose interaction graph has a bounded induced width [9] on which DR is polynomial. The idea is to find a small subset of conditioning variables, also called cutset, such that instantiating them leaves the remaining theory manageable for DR. [1]. The combined algorithm uses DP to instantiate the cutset variables and DR to resolve upon the remaining ones and its complexity is exponential in the sum of cutset size and the conditional induced-width.

Dividing the set of variables into cutset and resolution variables can be accomplished dynamically, during the algorithm's execution, or in advance of executing it. Accordingly, we consider two variations: Dynamic Conditioning + DR, called *DCDR*, and Static Conditioning + DR, called *SCDR*.

The rest of the paper has the following structure. Sections 2 and 3 contain the necessary definitions and preliminaries, section 4 discusses algorithms DCDR and SCDR, section 5 reports on our preliminary experimental results for solving propositional satisfiability. Concluding remarks are given in section 6.

2 Definitions

Propositional symbols, also called *variables*, are denoted by uppercase letters $P, Q, R, ...$; propositional literals (i.e., $P, \neg P$) by lowercase letters $p, q, r, ...$, and disjunctions of literals, or *clauses*, by $\alpha, \beta, ...$. A *unit clause* is a clause of size 1. A *formula*, or *theory*, φ in conjunctive normal form (cnf) is a conjunction of clauses. The set of propositional variables involved in φ is denoted as V. The notation $(\alpha \vee T)$ will be used as a shorthand for the disjunction $(P \vee Q \vee R \vee T)$, and $\alpha \vee \beta$ denotes the clause whose literal appears in either α or β. The *resolution* operation over two clauses $(\alpha \vee Q)$ and $(\beta \vee \neg Q)$ results in a clause $(\alpha \vee \beta)$, thus eliminating Q. *Unit resolution* is a resolution operation when one of the clauses is a unit clause. *Unit propagation* is a procedure that performs unit resolution on a given theory until no unit clauses are left. We denote by Q_i the positive assignment, or instantiation, $Q_i = true$, and by $\neg Q_i$ the negative instantiation, $Q_i = false$. $I(C)$ denotes an instantiation of a subset of variables $C \subseteq V$. The theory φ augmented with $I(C)$, $\varphi_{I(C)}$, is the *conditional restriction* of φ relative to $I(C)$.

The *interaction graph* of φ, denoted $G(\varphi)$, is an undirected graph that contains a node for each propositional variable and an arc connecting any two nodes whose associated variables appear in the same clause. Given an ordering d of the variables, the parent set of a node Q is the set of nodes connected to Q in $G(\varphi)$ that precede Q. The size of this parent set is called the *width* of Q relative to d. The width w_d of an ordering d is the maximum width of nodes along the ordering, and the width w of a graph is the minimal width over all its orderings. The graph generated by recursively connecting the parents of G, in the reverse order of d is called the *induced graph* of G, and is denoted by $IG_d(G)$. The width of $IG_d(G)$, $w^*(d)$, is the *induced width* of G [10, 8]. The *conditional interaction graph* w.r.t. $I(C)$, $G(\varphi_{I(C)})$, is obtained from $G(\varphi)$ by deleting [2] the nodes corresponding to variables in C.

[1] This is a generalization of the cycle-cutset algorithm proposed in [8] which transforms the interaction graph of a theory into a tree (CSPs whose graph is a tree are tractable) by instantiating the cutset variables.

[2] Deleting a node x from a graph $G(V, E)$ means removing x from V, and deleting all edges incident with x from E.

DP(φ)
Input: A cnf theory φ
Output: A decision of whether φ is satisfiable.
1. Unit_propagate(φ);
2. If the empty clause is generated, return($false$);
3. Else, if all variables are assigned, return($true$);
4. Else
5. $\quad\quad Q$ = some unassigned variable;
6. $\quad\quad$ return(**DP**($\varphi \wedge Q$) $\vee$
$\quad\quad\quad\quad\quad\quad\quad$ **DP**($\varphi \wedge \neg Q$))

Figure 1: The Davis-Putnam Procedure

The *conditional width* and *conditional induced width* of a theory φ relative to $I(C)$, denoted $w_{I(C)}$ and $w^*_{I(C)}$, are the width and induced width of $G(\varphi_{I(C)})$.

3 Preliminaries

The *Davis-Putnam Procedure (DP)* [4] and *Directional Resolution (DR)* [5, 9] are shown for completeness sake in figures 1 and 2, respectively. DP is the commonly used backtracking algorithm that systematically searches the space of possible truth assignments of propositional variables. DP performs unit propagation at each step, and is often augmented by dynamic variable ordering heuristic for selecting the next unassigned variable (line 5 in Figure 1).

DR [9] is an ordering-based restricted resolution that can be described using the notion of *buckets*. Given an ordering $d = Q_1, ..., Q_n$, all the clauses containing Q_i that do not contain any symbol higher in the ordering should be placed in $bucket_i$. Algorithm DR processes the buckets in a reverse order of d. When processing $bucket_i$, it resolves over Q_i all possible pairs of clauses in the bucket and inserts the resolvents into appropriate lower buckets. The output theory, $E_d(\varphi)$, is called the *directional extension* of φ. A theory has a non-empty directional extension (i.e. not containing the empty clause, $\emptyset$) iff it is satisfiable [5]. If $E_d(\varphi)$ is not empty, than any model of φ 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 $bucket_1$ (if the bucket is empty, assign Q_1 an arbitrary value); Step i. After assigning values to $Q_1, ..., Q_{i-1}$, assign a value to Q_i so that together with the previous assignments it will satisfy all clauses in $bucket_i$ [9]. It was shown that

Theorem 1: *[9] Given a theory φ and an ordering d, the time and space complexity of algorithm DR is $O(n \cdot exp(w^*(d)))$, where n is the number of propositional variables and $w^*(d)$ is the induced width of φ relative to d.*

In [9] we have shown that, although inefficient on uniformly generated problems, DR

DR(φ,d)
Input: A *cnf* theory φ, an ordering $d = Q_1, ..., Q_n$ of its variables.
Output: A decision of whether φ is satisfiable. If it is, a theory $E_d(\varphi)$, equivalent to φ, else an empty directional extension.
1. *Initialize:* generate an ordered partition of the clauses, $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the clauses whose highest literal is Q_i.
2. For $i = n$ to 1 do:
3. Resolve each pair $\{(\alpha \vee Q_i), (\beta \vee \neg Q_i)\} \subseteq bucket_i$. If $\gamma = \alpha \vee \beta$ is empty, return $E_d(\varphi) = \emptyset$, the theory is not satisfiable; else, determine the index of γ and add it to the appropriate bucket.
4. End-for.
5. Return $E_d(\varphi) \Longleftarrow \bigcup_i bucket_i$.

Figure 2: Algorithm *Directional Resolution (DR)*

greatly outperforms DP on problems with special structure, namely on *k-tree embeddings*. It is known that the induced width of a graph embedded in a k-tree is bounded by k [1]. A k-tree is defined recursively as follows. A clique of size k (complete graph with k vertices) is a k-tree. Given a k-tree defined on $Q_1, ..., Q_{i-1}$, a k-tree on $Q_1, ..., Q_i$ can be generated by selecting a clique of size k and connecting Q_i to every node in that clique.

Our earlier attempt to combine the advantages of DP on uniform problems with those of DR on structured problems resulted in the algorithm *Bounded DR + DP (BDR-DP)* which was shown empirically to be superior on both classes of problems. BDR-DP applies DR, restricted by the size of resolvents, as a preprocessing to DP. In this paper we present a more refined, approach to combining DR with DP.

4 Conditioning + Directional Resolution: Hybrid Algorithms

We combine backtracking with resolution, by splitting the variables into two subsets: *conditioning*, or *cutset*, variables, to be instantiated, and *resolution* variables, to be resolved upon. We propose two variations of splitting, static and dynamic, leading to two algorithms, *Dynamic Conditioning + DR*, called *DCDR*, and *Static Conditioning + DR*, called *SCDR*. We perform most of the experimental work on the dynamic version while use SCDR primarily for theoretical purposes.

A pseudocode of DCDR(b) is given in Figure 3. The algorithm takes as an input a propositional theory φ and determines whether φ is satisfiable. If it is, the algorithm returns a consistent partial assignment $I(C)$, and the conditional directional extension of φ, $E_d(\varphi_{I(C)})$. The algorithm decides dynamically whether the next variable is a cutset one or a resolution one, and processes the variable accordingly. DCDR(b) is described as a recursive procedure. It applies unit propagation to the current theory. If an inconsistency is discovered, the algorithm backtracks to the latest cutset variable (returns to the previous level of recursion

DCDR(φ, b)
Input: A cnf theory φ on the set of variables V.
Output: A decision of whether φ is satisfiable.
If it is, an assignment of cutset variables, $I(C)$, and $E_d(\varphi_{I(C)})$.
1. **If** Unit_propagate(φ) = *false*, return(*false*);
2. **Else If** no more variables to process, return *true*;
3. **While** $\exists Q$ s.t. $degree(Q) \leq b$
4. Resolve_Upon(Q);
5. **If** the empty clause is not generated,
6. add all resolvents to appropriate buckets of φ;
7. **Else** return *false*.
8. **End While**
11. Q = some unassigned variable;
12. $C = C \cup Q$;
13. return(DCDR($\varphi \wedge Q$, b) $\vee$
14. DCDR($\varphi \wedge \neg Q$, b)).

Figure 3: Algorithm DCDR

in line 12 or 13). Otherwise, it decides whether the next step is going to be resolution or conditioning (instantiation), and what variable it should be applied to. A variable is resolved upon if its conditional induced width is below b. Then, if selected, all the clauses in Q's bucket are resolved relative to Q, and the resolvents are added to the appropriate bucket. If Q is selected to be conditioning variable it is assigned a value, and DCDR(b) is called recursively. The algorithm returns the cutset C and the conditional directional extension $E_d(\varphi_{I(C)})$ along an ordering d, if φ is consistent.

The static version of the algorithm, SCDR(b), assumes a fixed ordering d, and a cutset C_b involving the first s_b variables in d. SCDR may save time by avoiding dynamic decision making; however, the partitioning of V into cutset and resolution variables may be less effective. Given a fixed ordering d, and a cutset size s_b such that the conditional induced width of the theory is below b, SCDR(b) applies DP to the subtheory restricted to C_b. If a consistent instantiation of the cutset variables is found, it processes the remaining theory by DR. If an empty clause is discovered, the algorithm backtrack,finds a new assignment to the cutset variables and continues.

We present some properties of the proposed family of algorithms described above.

Theorem 2: [soundness and completeness] *Algorithms DCDR(b) and SCDR(b) are sound and complete for satisfiability. If a theory φ is satisfiable, any model of φ that agrees with $I(C)$ can be generated in time $O(|E_d(\varphi_{I(C)})|)$ in a backtrack-free manner.*

Proof: As shown in [9], given a non-empty directional extension $E_d(\varphi)$ obtained by DR, any model of φ can be found in a backtrack-free manner in time $O(|E_d(\varphi)|)$ by a backtracking algorithm on the same ordering, but in opposite direction. Since the algorithms return the directional extension of $\varphi_{I(C)}$, the result follows.

$\square$

Theorem 3: [complexity] *The time complexity of algorithm SCDR(b) when applied to a theory φ is $O(exp(b+s_b))$, where s_b is a cutset size for which φ has conditional induced width bounded by b. The space complexity is $O(exp(b))$.*

Proof: Given an instantiation of the s_b cutset variables, the time and space complexity of DR in SCDR(b) is $O(exp(b))$. Enumerating all possible instantiations of the cutset variables takes $O(exp(s_b))$, yielding the result. $\square$

Similar result can be stated for DCDR(b), relative to the largest cutset size encountered during run-time. Note that using b as a parameter b allows control of the space complexity of the algorithms. We see that if $b \geq w^*(\varphi)$, the algorithm reduces to pure directional resolution, having time and space exponential in $w^*(\varphi)$ complexity. If $b \leq 1$, the algorithm is close to pure DP, whose time complexity becomes $O(exp(s_1))$, while its space complexity is linear. It is known that the best *cycle cutset* of a theory, s_0, and the smallest induced width, w^*, obey the relation $w^* \leq s_0 + 1$ [2]. Thus the time bound of the parameterized algorithms are smaller as b increases. When we use intermediate values of b, we activate an hybrid algorithm that trade space for time; as b increases, the algorithm may require more space and less time.

Different heuristic orderings can be used in both the static and the dynamic versions of the algorithm and they may affect the results dramatically. In both cases we use the following principle: select Q as resolution variable if its conditional width is below b. Select Q as a next conditioning variable if it has the maximal degree in the current graph.

Note that in case of SCDR(b) the upper bound b on $w^*_{I(C)}$ may be very loose; at run time some of the resolution variables might be instantiated by unit propagation. Indeed, our experiments show that dynamic ordering provides a tighter upper bound on the effective $w^*_{I(C)}$.

5 Experimental Results

5.1 Methodology

We compare algorithms DCDR(b) for different values of bound b. With SCDR(b) we performed just a limited amount of experiments. We also compare those algorithms with pure DP. Our implementation of DP uses the *2-literal clause heuristic* proposed in [3]. The same heuristic is used when processing cutset variables in $DCDR(b)$ and SCDR(b).

Several random problem generators are used. *Uniform k-cnfs* are obtained using the generator proposed in [12]. It takes as an input the number of variables n, the number of clauses m, and the number of literals per clause k, and obtains each clause by choosing k variables randomly from the set of n variables and by determining the polarity of each literal with probability $p = 0.5$.

To test our algorithms on problems having bounded w^* we implemented a (k, m)-*trees* generator, which generalizes the idea of k-trees [1]. A (k, m)-tree is a tree of cliques, each having $(k + m)$ nodes, where k is the size of intersection between each two neighboring cliques. Thus, conventional k-trees are $(k, 1)$-trees. The (k, m)-trees generator takes as an input k, m; the number of cliques, *Ncliques*; and the number of clauses for each clique, *Nclauses*. Given a set of cliques previously generated, it chooses a clique, randomly selects k variables in this clique, adds m new variables, and generates *Nclauses* clauses on the new clique. Note that *Nclauses* is *not* the number of clauses per each clique, because each new clique shares variables with other cliques. The induced width of a (k, m)-tree is bounded by $k + m - 1$.

For each algorithm, we computed the CPU time, the number of dead ends encountered, the cutset size, the number of variables resolved upon, the number of new clauses in the output theory, and the total number of new clauses generated.

We tested problem instances in the *phase transition* region which tends to yield difficult problems. According to experimental studies, on uniform 3-cnfs, the crossover point in such a transition phase occurs when the clauses/variables ratio equals approximately 4.3 [12, 3]. For (k, m)-trees, the transition region is not identified yet, and we had to experiment with several combinations of parameters in order to find relatively hard instances.

5.2 Experimental results for DCDR(b)

Our results are summarized in Figures 4-7. In Figure 5, we focus on uniform 3-cnfs having 100 variables and 400 clauses. In Figures 6 and 7 we provide the same statistics when experimenting with $(4, 5)$-trees and $(4, 8)$-trees, respectively. For each class we show the average time, deadends and number of clauses generated. The averages for uniform 3-cnfs are computed on 100 problem instances. We experimented with relatively few (k, m)-tree instances while trying many different combinations of parameters. An enormous amount of time is needed to cover all algorithms with various bounds over large number of regions in a statistically significant way. We therefore view our results as preliminary. Many more instances need to be tested before we can have conclusive results. Nevertheless, we feel that our preliminary results indicate the general promise of the approach.

As expected, the performance of DCDR(b) depends on the induced width of the theories tested. We have observed three different patterns of the algorithm's behavior as a function of $w*$. Figure 7 describes the average number of resolution variables in each of the three classes and may hint at the potential of these algorithms for knowledge compilation. When the cutset is small, many variables are resolved upon, so the resulting conditional directional extension codes a larger portion of the solutions, all agreeing with the assignment to the cutset variables.

- On problems having large w^*, such as uniform 3-cnfs in the transition region (see Figure 4), the time complexity of DCDR(b) is similar to DCDR(-1) when b is small. However, when b increases, the CPU time grows exponentially. Indeed, the number of dead ends encountered by DCDR(b) is decreasing too slowly relative to the exponential

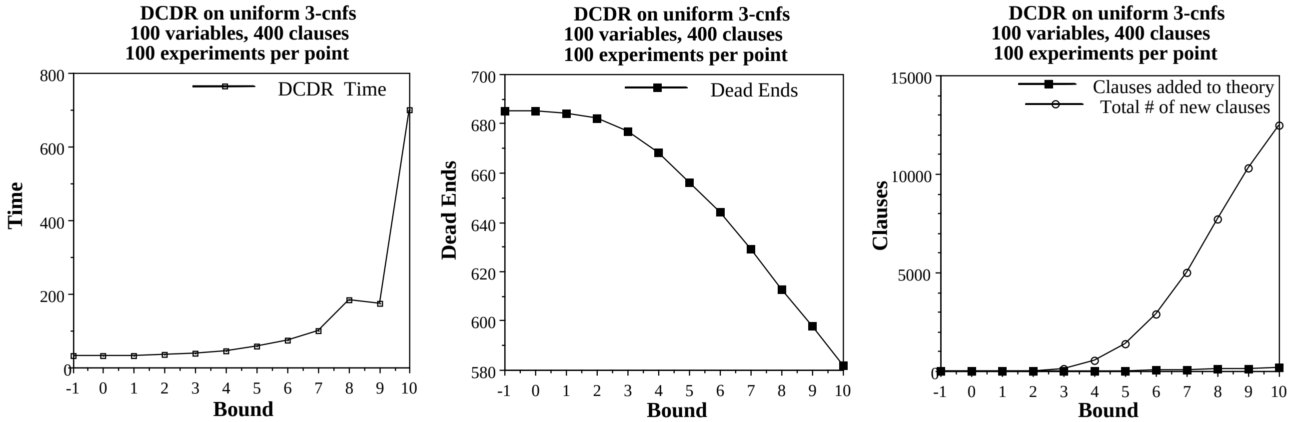

Figure 4: DCDR on uniform 3-cnfs: time, the number of deadends, total number of new clauses

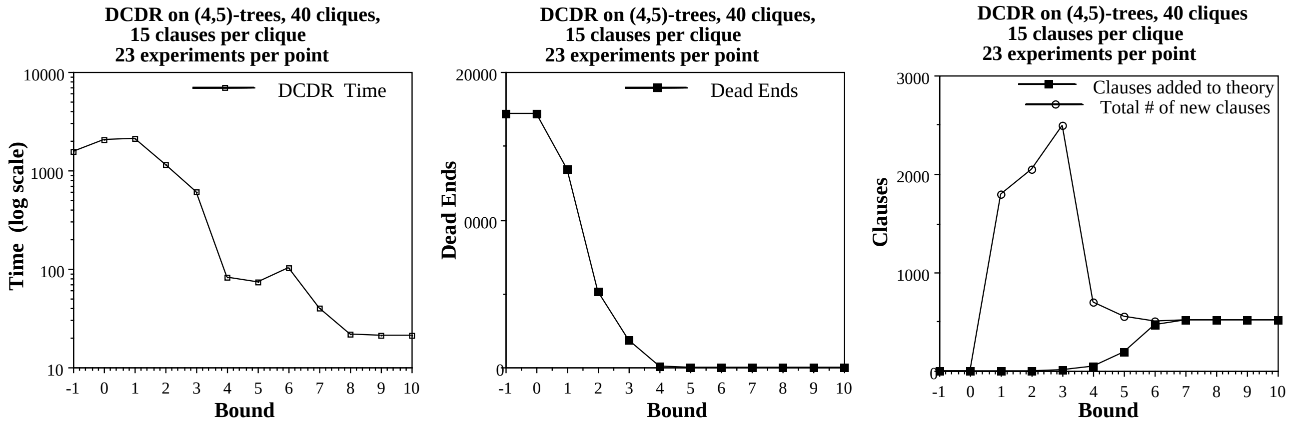

Figure 5: DCDR on (4,5)-trees: time, the number of deadends, total number of new clauses

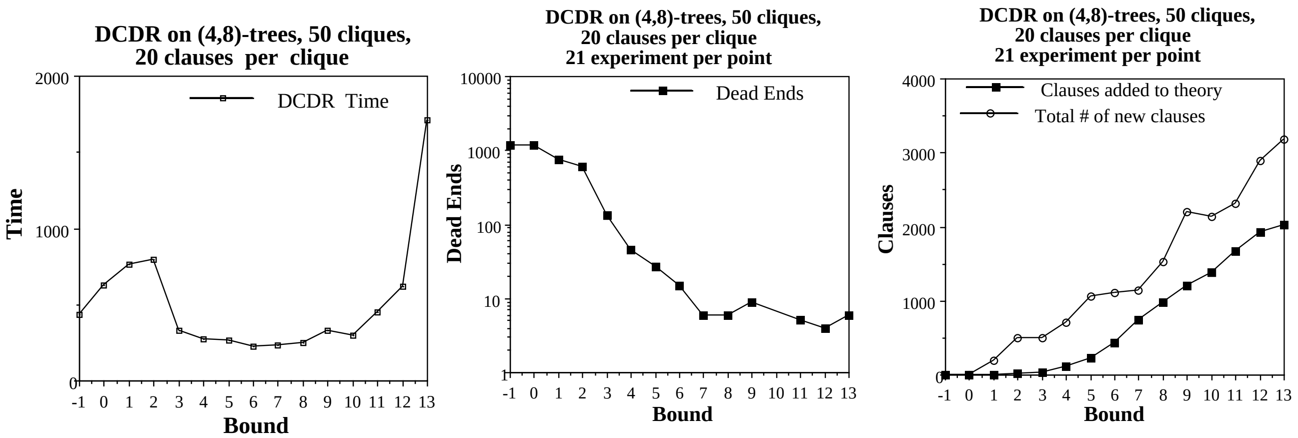

Figure 6: DCDR on (4,8)-trees: time, the number of deadends, total number of new clauses

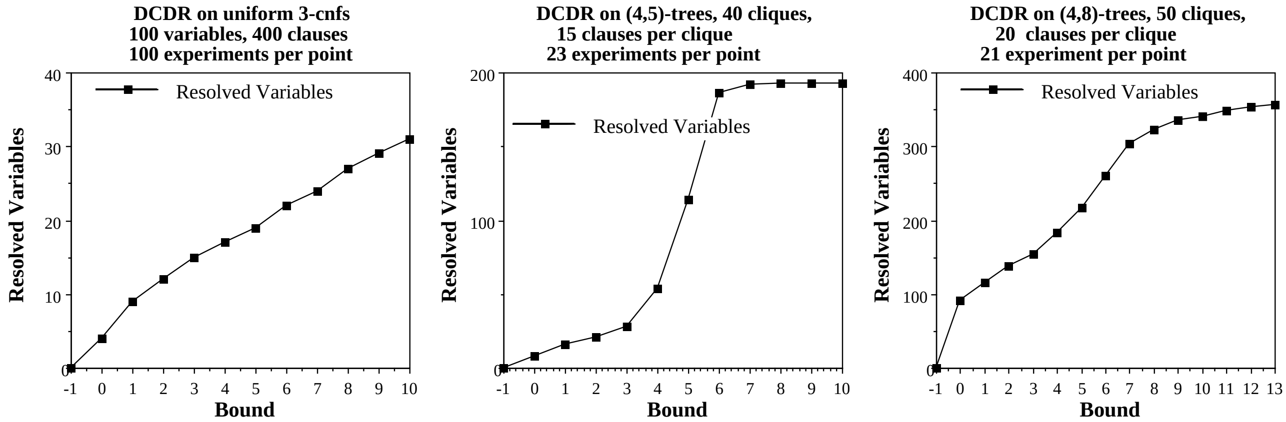

Figure 7: DCDR: the number of resolved variables on different problems

(in b) growth in the total number of generated clauses. Note that the number of new clauses in the final theory grows in a slower pace, and, therefore, the final conditional directional extensions have manageable sizes. When looking at the data in Figure 7, we see that for relatively large b's the cutset size takes at most 70% of the variables. We have obtained similar results when experimenting with uniform theories having 150 variables and 640 clauses.

- Clearly, DR is equivalent to DCDR(b) whenever b is equal or greater then $w*$. Therefore, for theories having small induced width, DCDR(b) coincides with DR for relatively small values of b. Figure 5 demonstrates this behavior on (4,5)-trees with 40 cliques, 15 clauses per clique, and induced width 6. A similar picture was observed on (3,6)-trees having 40 cliques, 15 clauses per clique, and (2,5)-trees having 40 cliques, 14 clauses per clique. We see that starting at $b \geq 8$, the CPU time, the total number of clauses generated, and the number of new clauses added to the output theory, are identical. Naturally, we observe here the same phenomenon when comparing $DCDR(-1)$ to $DCDR$(b) with $b \geq w^*$ as we observed in [9] comparing DP and DR. Note that for very small values of b ($b = 0, 1, 2, 3$), the efficiency of DCDR(b) was sometimes worse than that of DCDR(-1) due to the overhead incurred by extra clause generation.

- When testing DCDR(b) on (k, m)-trees with larger size of cliques (Figure 6), we observed an intermediate region for b's values that yielded a better performance than both extremes. This happens when DP (or, equivalently, DCDR(-1)) is still inefficient on the structured problems, while at the same time large induced width makes pure DR too costly time- and space-wise.

  As we see in Figure 6, for (4,8)-trees, the optimal values of b appear between 5 and 8. Similar pattern has been observed on (4,7)-trees with 50 cliques and 18 to 20 clauses per clique. In those cases a substantial number of the variables are resolved upon, resulting in an effective conditional directional extension that captures many solutions.

Finally, in Figure 8 we summarize the results for DCDR(-1), DCDR(5), and DCDR(13)

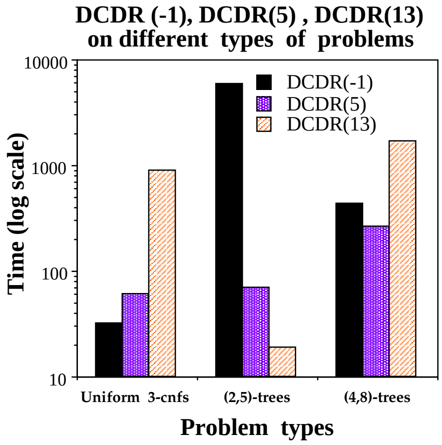

Figure 8: Relative performance of DCDR(b) for $b = -1, 5, 13$ on different types of problems

on all three classes of problems considered in the experiments. These results suggest that intermediate bounds, like 5, can be effective in all three cases.

5.3 Results for SCDR

When comparing DCDR(-1) with pure DP, we observed that DP was significantly more efficient, although the two algorithms are supposedly identical. This is due to the not very efficient implementation of DCDR(b), which we hope to improve in the future. Also, we expected that SCDR(b) will avoid this overhead. Indeed, the CPU time of SCDR(-1) was comparable to the DP's time, which was an order of magnitude better than the time for DCDR(-1). Although we have experimented with SCDR, we omit any tables and figures since the results are preliminary. The main difference observed between DCDR(b) and SCDR(b) is that for small values of b static version is more efficient due to low overhead. On the other hand, since b here provides a looser upper bound on w^*, larger values of b are necessary in order to observe a transition a from DP to DR.

6 Summary and conclusions

We have presented a family of hybrid algorithms that combine backtracking search (e.g. Davis-Putnam procedure) with resolution (e.g. directional resolution). Backtracking is a form of conditioning, while resolution is a variable elimination procedure. Conditioning and elimination algorithms are widely used in solving constraint satisfaction, constraint optimization and in reasoning under uncertainty. The combined algorithms are parameterized by a constant b, which bounds the amount of resolution permitted, and which invokes the notion of *induced width*. Given b, the algorithm selects a subset of conditioning variables C_b such that the resulting (conditional) theory has an induced width not exceeding b. The hybrid algorithm searches the space of truth assignments for the conditioning variables and resolves upon the rest of the variables.

This family of algorithms lies between the extreme points of DP and DR strategies, and enables us to exploit the virtues of both on given problem instance. In particular we have shown that 1. hybrid algorithms can be more effective than either backtracking (DP) or resolution (DR) on some classes of problems having intermediate sizes of induced width, 2. hybrid algorithms require less space than pure resolution, and 3. hybrid algorithms can output a compiled theory from which a portion of the solution set can be generated in linear time.

Our preliminary experimental results show that the induced width provides a reasonable predictor of the level b that should be used for a given instance. Roughly speaking, when $w*$ is very large we choose $b \leq 1$, when the induced width is very small (less than 4) we choose large b's, while for intermediate levels of $w*$ it is better to choose a bounded level of b. We are not ready at this point to suggest a more refined recommendation. However, from our preliminary study we can see that when using the hybrid algorithms with b ranging over $b = 5, ..., 8$, performance improved over all problem classes tested. These algorithms automatically reduce to DP on uniform instances, to DR for (k, m)-trees having small width, while on intermediate sizes of w* the algorithms exploit both the DP and DR strategies in a way that outperforms each of them.

References

[1] Arnborg, S., Corneil, D.G., and Proskurowski, A., Complexity of Finding Embedding in a k-tree, *Journal of SIAM, Algebraic Discrete Methods*, 8(2):177-184 (1987).

[2] Bertele, U. and Brioschi, F., *Nonserial Dynamic Programming*, Academic Press, New York, 1972.

[3] Crawford, J.M. and Auton, L.D., Experimental Results on the Cross-over Point in Satisfiability Problems, in *Proceedings of the National Conference on Artificial Intelligence (AAAI-93)* , July 1993, pp. 21-27.

[4] M. Davis, G. Logemann and D. Loveland, A machine program for theorem proving, *Communications of the ACM*, 5, 1962, pp. 394-397.

[5] M. Davis and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM*, 7, 1960, pp. 201-215.

[6] R. Dechter, Constraint networks. *Encyclopedia of Artificial Intelligence* (2nd Ed.), John Wiley, New York, 1991 pp. 276-285.

[7] R. Dechter, Topological parameters for time-space tradeoff, In *Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics, AI/MATH-96*, 1996.

[8] R. Dechter and J. Pearl, Network-based heuristics for constraint satisfaction problems. In *Artificial Intelligence*, 34, pp. 1-38, 1987.

[9] R. Dechter and I. Rish, Directional Resolution: The Davis-Putnam Procedure, Revisited. In *Proceedings of KR-94*, pp. 134-145, 1994.

[10] E.C. Freuder, A sufficient condition for backtrack-free search. *Journal of the ACM*, 29, 1982, 24-32.

[11] K. Kask and R. Dechter, GSAT and Local Consistency. In *Proceedings of IJCAI-95*, pp. 616-622, 1995.

[12] Mitchell, D., Selman, B., and Levesque, H., Hard and Easy Distributions of SAT Problems, in *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA, July 1992, pp. 459-465.

[13] Selman, B., Levesque, H., and Mitchell, D., A New Method for Solving Hard Satisfiability Problems, in *Proceedings of the National Conference on Artificial Intelligence (AAAI-92)*, San Jose, CA, July 1992, pp. 440-446.

[14] B. Smith and S. Grant, Sparse Constraint Graphs and Exceptionally Hard Problems. In *Proceedings of IJCAI-95*, pp.646-651, 1995.