Resolution versus Search: Two Strategies for SAT [*]

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

Abstract

The paper compares two popular strategies for solving propositional satisfiability, *backtracking search* and *resolution*, and analyzes the complexity of a directional resolution algorithm (DR) as a function of the "width" (w^*) of the problem's graph. Our empirical evaluation confirms theoretical prediction, showing that on low-w^* problems DR is very efficient, greatly outperforming the backtracking-based Davis-Putnam-Logemann-Loveland procedure (DP). We also emphasize the knowledge-compilation properties of DR and extend it to a tree-clustering algorithm that facilitates query answering. Finally, we propose two hybrid algorithms that combine the advantages of both DR and DP. These algorithms use control parameters that bound the complexity of resolution and allow time/space trade-offs that can be adjusted to the problem structure and to the user's computational resources. Empirical studies demonstrate the advantages of such hybrid schemes.

Keywords: propositional satisfiability, backtracking search, resolution, computational complexity, knowledge compilation, empirical studies.

[*]This work was partially supported by NSF grant IRI-9157636.

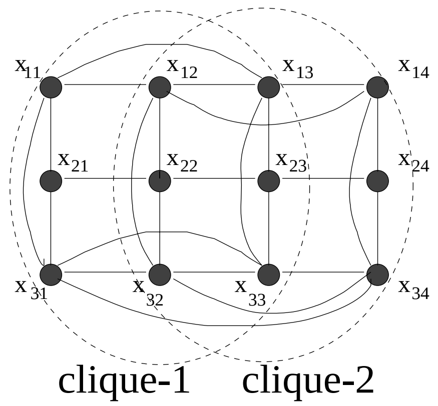
1 Introduction

Propositional satisfiability (SAT) is a prototypical example of an NP-complete problem; any NP-complete problem is reducible to SAT in polynomial time [8]. Since many practical applications such as planning, scheduling, and diagnosis can be formulated as propositional satisfiability, finding algorithms with good average performance has been a focus of extensive research for many years [59, 10, 34, 45, 46, 3]. In this paper, we consider *complete* SAT algorithms that can always determine satisfiability as opposed to incomplete *local search* techniques [59, 58]. The two most widely used complete techniques are backtracking search (e.g., the Davis-Putnam Procedure [11]) and resolution (e.g., Directional Resolution [12, 23]). We compare both approaches theoretically and empirically, suggesting several ways of combining them into more effective hybrid algorithms.

In 1960, Davis and Putnam presented a resolution algorithm for deciding propositional satisfiability (the Davis-Putnam algorithm [12]). They proved that a restricted amount of resolution performed along some ordering of the propositions in a propositional theory is sufficient for deciding satisfiability. However, this algorithm has received limited attention and analyses of its performance have emphasized its worst-case exponential behavior [35, 39], while overlooking its virtues. It was quickly overshadowed by the Davis-Putnam Procedure, introduced in 1962 by Davis, Logemann, and Loveland [11]. They proposed a minor syntactic modification of the original algorithm: the resolution rule was replaced by a splitting rule in order to avoid an exponential memory explosion. However, this modification changed the nature of the algorithm and transformed it into a backtracking scheme. Most of the work on propositional satisfiability quotes the backtracking version [40, 49]. We will refer to the original Davis-Putnam algorithm as *DP-resolution*, or *directional resolution (DR)* [1], and to its later modification as *DP-backtracking*, or *DP* (also called DPLL in the SAT community).

Our evaluation has a substantial empirical component. A common approach used in the empirical SAT community is to test algorithms on randomly generated problems, such as uniform random k-SAT [49]. However, these benchmarks often fail to simulate realistic problems. On the other hand, "real-life" benchmarks are often available only on an instance-by-instance basis without any knowledge of underlying distributions which makes the empirical results hard to generalize. An alternative approach is to use *structured* random problem generators inspired by the properties of some realistic domains. For example, Figure 1 illustrates the *unit commitment* problem of scheduling a set of n power generating units over T hours (here $n = 3$ and $T = 4$). The state of unit i at time t ("up"

[1]A similar approach known as "ordered resolution" can be viewed as a more sophisticated first order version of directional resolution [25].



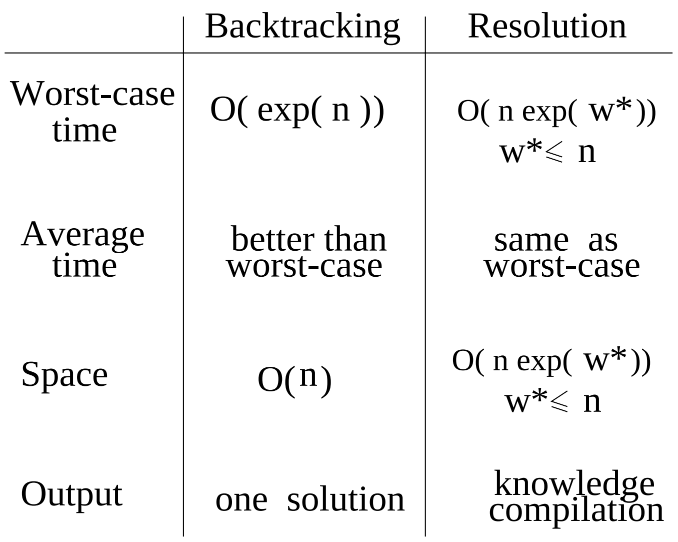
Unit #	Min Up Time	Min Down Time
1	3	2
2	2	1
3	4	1

Figure 1: An example of a "temporal chain": the unit commitment problem for 3 units over 4 hours.

or "down") is specified by the value of boolean variable x_{it} (0 or 1), while the minimum up- and down-time constraints specify how long a unit must stay in a particular state before it can be switched. The corresponding constraint graph can be embedded in a chain of cliques where each clique includes the variables within the given number of time slices determined by the up- and down-time constraints. These clique-chain structures are common in many temporal domains that possess the *Markov property* (the future is independent of the past given the present). Another example of structured domain is circuit diagnosis. In [27] it was shown that circuit-diagnosis benchmarks can be embedded in a tree of cliques, where the clique sizes are substantially smaller than the overall number of variables. In general, one can imagine a variety of real-life domains having such structure that is captured by *k-tree-embeddings* [1] used in our random problem generators.

Our empirical studies of SAT algorithms confirm previous results: DR is very inefficient when dealing with unstructured uniform random problems. However, on structured problems such as k-tree embeddings having bounded *induced width*, directional resolution outperforms DP-backtracking by several orders of magnitude. The induced width (denoted w^*) is a graph parameter that describes the size of the largest clique created in the problem's interaction graph during inference. We show that the worst-case time and space complexity of DR is $O(n \cdot exp(w^*))$, where n is the number of variables. We also identify tractable problem classes based on a more refined syntactic parameter, called *diversity*.

Since the induced width is often smaller than the number of propositional variables, n, DR's worst-case bound is generally better than $O(exp(n))$, the worst-case time bound for DP. In practice, however, DP-backtracking – one of the best complete SAT algorithms



	Backtracking	Resolution
Worst-case time	O(exp(n))	O(n exp(w*)) w*⩽ n
Average time	better than worst-case	same as worst-case
Space	O(n)	O(n exp(w*)) w*⩽ n
Output	one solution	knowledge compilation

Figure 2: Comparison between backtracking and resolution.

available – is often much more efficient than its worst-case bound. It demonstrates "great discrepancies in execution time" (D.E. Knuth), encountering rare but exceptionally hard problems [60]. Recent studies suggest that the empirical performance of backtracking algorithms can be modeled by long-tail exponential-family distributions, such as lognormal and Weibull [32, 54]. The average complexity of algorithm DR, on the other hand, is close to its worst-case [18]. It is important to note that the space complexity of DP is $O(n)$, while DR is space-exponential in w^*. Another difference is that in addition to deciding satisfiability and finding a solution (a model), directional resolution also generates an equivalent theory that allows finding each model in linear time (and finding all models in time linear in the number of models), and thus can be viewed as a knowledge-compilation algorithm.

The complementary characteristics of backtracking and resolution (Figure 2) call for hybrid algorithms. We present two hybrid schemes, both using control parameters that restrict the amount of resolution by bounding the resolvent size, either in a preprocessing phase or dynamically during search. These parameters allow time/space trade-offs that can be adjusted to the given problem structure and to the computational resources. Empirical studies demonstrate the advantages of these flexible hybrid schemes over both extremes, backtracking and resolution.

This paper is an extension of the work presented in [23] and includes several new results. A tree-clustering algorithm for query processing that extends DR is presented and analyzed. The bounded directional resolution (BDR) approach proposed in [23] is subjected to a much more extensive empirical tests that include both randomly generated problems and DIMACS benchmarks. Finally, a new hybrid algorithm, *DCDR*, is introduced and evaluated empirically on a variety of problems.

The rest of this paper is organized as follows. Section 2 provides necessary definitions. Section 3 describes directional resolution (DR), our version of the original Davis-Putnam algorithm expressed within the *bucket-elimination* framework. Section 4 discusses the complexity of DR and identifies tractable classes. An extension of DR to tree-clustering scheme is presented in Section 5, while Section 6 focuses on DP-backtracking. Empirical comparison of DR and DP is presented in Section 7. Section 8 introduces the two hybrid schemes, BDR-DP and DCDR, and empirically evaluates their effectiveness. Related work and conclusions are discussed in Sections 9 and 10. Proofs of theorems are given in the Appendix A.

2 Definition and Preliminaries

We denote propositional variables, or propositions, by uppercase letters, e.g. P, Q, R, propositional literals (propositions or their negations, such as P and $\neg P$) by lowercase letters, e.g., p, q, r, and disjunctions of literals, or *clauses*, by the letters of the Greek alphabet, e.g., α, β, γ. For instance, $\alpha = (P \vee Q \vee R)$ is a clause. We will sometimes denote the clause $(P \vee Q \vee R)$ by $\{P, Q, R\}$. A *unit clause* is a clause with only one literal. A clause is *positive* if it contains only positive literals and is *negative* if it contains only negative literals. The notation $(\alpha \vee T)$ is used as shorthand for $(P \vee Q \vee R \vee T)$, while $\alpha \vee \beta$ refers to the clause whose literals appear in either α or β. A clause β is *subsumed* by a clause α if β's literals include all α's literals. A clause is a *tautology*, if for some proposition Q the clause includes both Q and $\neg Q$. A *propositional theory* φ in conjunctive normal form (*cnf*) is represented as a set $\{\alpha_1, ..., \alpha_t\}$ denoting the conjunction of clauses $\alpha_1, ..., \alpha_t$. A *k-cnf theory* contains only clauses of length k or less. A propositional cnf theory φ defined on a set of n variables Q_1,...,Q_n is often called simply "a theory φ".

The set of *models* of a theory φ is the set of all truth assignments to its variables that satisfy φ. A clause α is *entailed* by φ (denoted $\varphi \models \alpha$), if and only if α is true in all models of φ. A propositional satisfiability problem (SAT) is to decide whether a given cnf theory has a model. A SAT problem defined on k-cnfs is called a *k-SAT* problem.

The structure of a propositional theory can be described by an *interaction graph.* The interaction graph of a propositional theory φ, denoted $G(\varphi)$, is an undirected graph that contains a node for each propositional variable and an edge for each pair of nodes that correspond to variables appearing in the same clause. For example, the interaction graph of theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$ is shown in Figure 3a.

One commonly used approach to satisfiability testing is based on the *resolution* op-

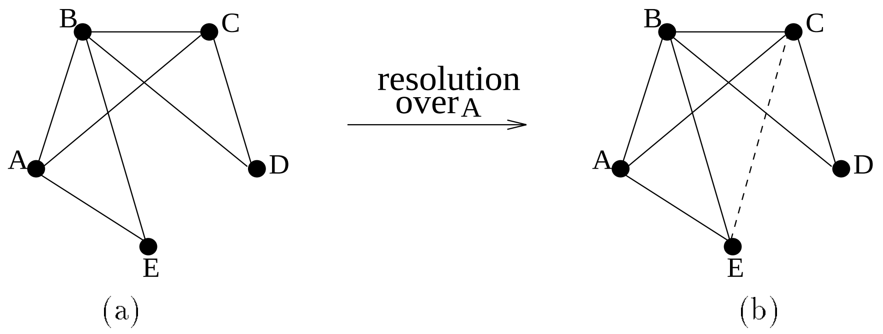


Figure 3: (a) The interaction graph of theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$, and (b) the effect of resolution over A on that graph.

eration. Resolution over two clauses $(\alpha \vee Q)$ and $(\beta \vee \neg Q)$ results in a clause $(\alpha \vee \beta)$ (called *resolvent*) eliminating variable Q. The interaction graph of a theory processed by resolution should be augmented with new edges reflecting the added resolvents. For example, resolution over variable A in φ_1 generates a new clause $(B \vee C \vee E)$, so the graph of the resulting theory has an edge between nodes E and C as shown in Figure 3b. Resolution with a unit clause is called *unit resolution. Unit propagation* is an algorithm that applies unit resolution to a given cnf theory until no new clauses can be deduced.

Propositional satisfiability is a special case of *constraint satisfaction problem (CSP).* CSP is defined on a *constraint network* $< X, D, C >$, where $X = \{X_1, ..., X_n\}$ is the set of *variables*, associated with a set of finite *domains*, $D = \{D_i, ..., D_n\}$, and a set of *constraints*, $C = \{C_1, ..., C_m\}$. Each constraint C_i is a relation $R_i \subseteq D_{i_1} \times ... \times D_{i_k}$ defined on a subset of variables $S_i = \{X_{i_1}, ..., X_{i_k}\}$. A constraint network can be associated with an undirected *constraint graph* where nodes correspond to variables and two nodes are connected if and only if they participate in the same constraint. The *constraint satisfaction problem (CSP)* is to find a value assignment to all the variables (called a *solution*) that is consistent with all the constraints. If no such assignment exists, the network is *inconsistent.* A constraint network is *binary* if each constraint is defined on at most two variables.

3 Directional Resolution (DR)

DP-resolution [12] is an ordering-based resolution algorithm that can be described as follows. Given an arbitrary ordering of the propositional variables, we assign to each clause the index of its highest literal in the ordering. Then resolution is applied only to clauses having the same index and only on their highest literal. The result of this restriction is a systematic elimination of literals from the set of clauses that are candidates

```
**Directional Resolution: DR**
Input: A cnf theory $\varphi$, $o = Q_1, ..., Q_n$.
Output: The decision of whether $\varphi$ is satisfiable.
If it is, the directional extension $E_o(\varphi)$ equivalent to $\varphi$.
1. Initialize: generate a partition of 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:
        If there is a unit clause in $bucket_i$,
        do unit resolution in $bucket_i$
        else resolve each pair $\{(\alpha \vee Q_i), (\beta \vee \neg Q_i)\} \subseteq bucket_i$.
        If $\gamma = \alpha \vee \beta$ is empty, return "$\varphi$ is unsatisfiable"
        else add $\gamma$ to the bucket of its highest variable.
3. Return "$\varphi$ is satisfiable" and $E_o(\varphi) = \bigcup_i bucket_i$.
```

Figure 4: Algorithm Directional Resolution (DR).

for future resolution. The original DP-resolution also includes two additional steps, one forcing unit resolution whenever possible, and one assigning values to *all-positive* and *all-negative* variables. An all-positive (all-negative) variable is a variable that appears only positively (negatively) in a given theory, so that assigning such a variable the value "true" ("false") is equivalent to deleting all relevant clauses from the theory. There are other intermediate steps that can be introduced between the basic steps of eliminating the highest indexed variable, such as deleting subsumed clauses. Albeit, we will focus on the ordered elimination step and refer to auxiliary steps only when necessary. We are interested not only in deciding satisfiability but in the set of clauses accumulated by this process constituting an equivalent theory with useful computational features. Algorithm *directional resolution* (DR), the core of DP-resolution, is presented in Figure 4. This algorithm can be described using the notion of *buckets*, which define an ordered partitioning of clauses in φ, as follows. Given an ordering $o = (Q_1, ..., Q_n)$ of the variables in φ, all the clauses containing Q_i that do not contain any symbol higher in the ordering are placed in $bucket_i$. The algorithm processes the buckets in a reverse order of o, from Q_n to Q_1. Processing $bucket_i$ involves resolving over Q_i all possible pairs of clauses in that bucket. Each resolvent is added to the bucket of its highest variable Q_j (clearly, $j < i$). Note that if the bucket contains a unit clause (Q_i or $\neg Q_i$), only unit resolutions are performed. Clearly, a useful dynamic-order heuristic (not included in our current implementation) is to processes next a bucket with a unit clause. The output theory,

```
find-model ($E_o(\varphi), o$ )
Input: A directional extension $E_o(\varphi)$, $o = Q_1, ..., Q_n$.
Output: A model of $\varphi$.
1. For $i = 1$ to N
        $Q_i \leftarrow$ a value $q_i$ consistent with the assignment to
        $Q_1, ..., Q_{i-1}$ and with all the clauses in $bucket_i$.
2. Return $q_1, ..., q_n$.
```

Figure 5: Algorithm *find-model*.

$E_o(\varphi)$, is called the *directional extension* of φ along o. As shown by Davis and Putnam [12], the algorithm finds a satisfying assignment to a given theory if and only if there exists one. Namely,

Theorem 1: *[12] Algorithm DR is sound and complete.* $\Box$

A model of a theory φ can be easily found by consulting $E_o(\varphi)$ using a simple model-generating procedure *find-model* in Figure 5. Formally,

Theorem 2: (model generation)
Given $E_o(\varphi)$ of a satisfiable theory φ, the procedure find-model *generates a model of φ backtrack-free, in time $O(|E_o(\varphi)|)$.* $\Box$

Example 1: Given the input theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$, and an ordering $o = (E, D, C, B, A)$, the theory is partitioned into buckets and processed by directional resolution in reverse order[2]. Resolving over variable A produces a new clause $(B \vee C \vee E)$, which is placed in $bucket_B$. Resolving over B then produces clause $(C \vee D \vee E)$ which is placed in $bucket_C$. Finally, resolving over C produces clause $(D \vee E)$ which is placed in $bucket_D$. Directional resolution now terminates, since no resolution can be performed in $bucket_D$ and $bucket_E$. The output is a non-empty directional extension $E_o(\varphi_1)$. Once the directional extension is available, model generation begins. There are no clauses in the bucket of E, the first variable in the ordering, and therefore E can be assigned any value (e.g., $E = 0$). Given $E = 0$, the clause $(D \vee E)$ in $bucket_D$ implies $D = 1$, clause $\neg C$ in $bucket_C$ implies $C = 0$, and clause $(B \vee C \vee E)$ in $bucket_B$, together

[2]For illustration, we selected an arbitrary ordering which is not the most efficient one. Variable ordering heuristics will be discussed in Section 4.3.

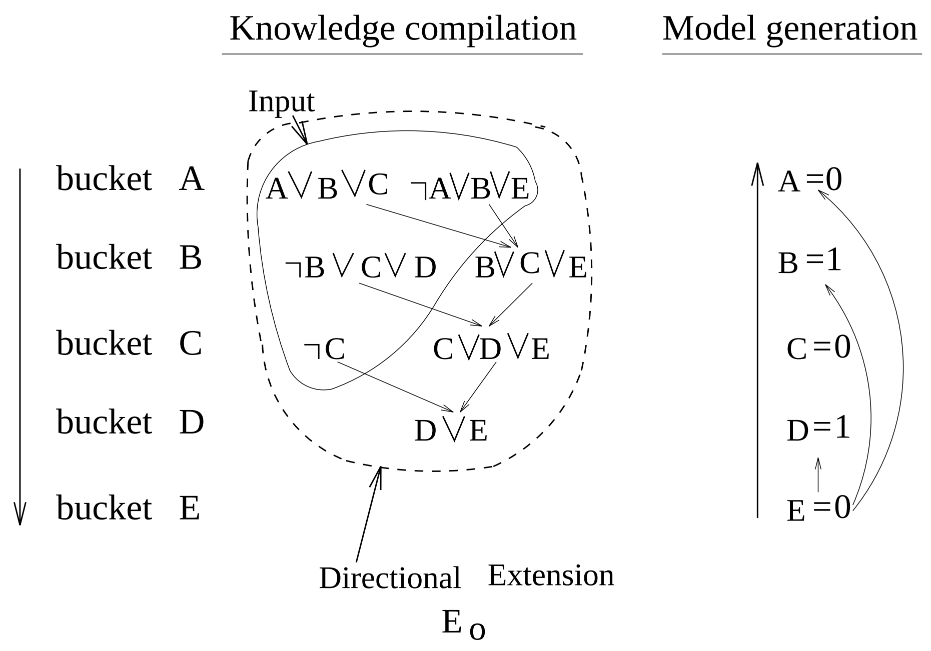


Figure 6: A trace of algorithm DR on the theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$ along the ordering $o = (E, D, C, B, A)$.

with the current assignments to C and E, implies $B = 1$. Finally, A can be assigned any value since both clauses in its bucket are satisfied by previous assignments.

As stated in Theorem 2, given a directional extension, a model can be generated in linear time. Once $E_o(\varphi)$ is compiled, determining the entailment of a single literal requires checking the bucket of that literal first. If the literal appears there as a unit clause, it is entailed; if it is not entailed, its negation is added to the appropriate bucket and the algorithm resumes from that bucket. If the empty clause is generated, the literal is entailed. Entailment queries will also be discussed in Section 5.

4 Complexity and Tractability

Clearly, the effectiveness of algorithm DR depends on the the size of its output theory $E_o(\varphi)$.

Theorem 3: (complexity)
Given a theory φ and an ordering o, the time complexity of algorithm DR is $O(n \cdot |E_o(\varphi)|^2)$ where n is the number of variables. $\square$

The size of the directional extension and therefore the complexity of directional resolution is worst-case exponential in the number of variables. However, there are identifiable cases when the size of $E_o(\varphi)$ is bounded, yielding tractable problem classes. The order of variable processing has a particularly significant effect on the size of the directional extension. Consider the following two examples:

Example 2: Let $\varphi_2 = \{(B \vee A), (C \vee \neg A), (D \vee A), (E \vee \neg A)\}$. Given the ordering $o_1 = (E, B, C, D, A)$, all clauses are initially placed in $bucket(A)$. Applying DR along the (reverse) ordering, we get: $bucket(D) = \{(C \vee D), (D \vee E)\}$, $bucket(C) = \{(B \vee C)\}$, $bucket(B) = \{(B \vee E)\}$. In contrast, the directional extension along ordering $o_2 = (A, B, C, D, E)$ is identical to the input theory φ_2 since each bucket contains at most one clause.

Example 3: Consider the theory $\varphi_3 = \{(\neg A \vee B), (A \vee \neg C), (\neg B \vee D), (C \vee D \vee E)\}$. The directional extensions of φ_3 along ordering $o_1 = (A, B, C, D, E)$ and $o_2 = (D, E, C, B, A)$ are $E_{o_1}(\varphi_3) = \varphi_3$ and $E_{o_2}(\varphi_3) = \varphi_3 \cup \{(B \vee \neg C), (\neg C \vee D), (E \vee D)\}$, respectively.

In example 2, variable A appears in all clauses. Therefore, it can potentially generate new clauses when resolved upon, unless it is processed last (i.e., it appears first in the ordering), as in o_2. This shows that the interactions among variables can affect the performance of the algorithm and should be consulted for producing preferred orderings. In example 3, on the other hand, all the symbols have the same type of interaction, each (except E) appearing in two clauses. Nevertheless, D appears positive in both clauses in its bucket, therefore, it will not be resolved upon and can be processed first. Subsequently, B and C appear only negatively in the remaining theory and will not add new clauses. Inspired by these two examples, we will now provide a connection between the algorithm's complexity and two parameters: a topological parameter, called *induced width*, and a syntactic parameter, called *diversity*.

4.1 Induced width

In this section we show that the size of the directional extension and therefore the complexity of directional resolution can be estimated using a graph parameter called *induced width*.

As noted before, DR creates new clauses which correspond to new edges in the resulting interaction graph (we say that DR "induces" new edges). Figure 7 illustrates again the performance of directional resolution on theory φ_1 along ordering $o = (E, D, C, B, A)$, showing this time the interaction graph of $E_o(\varphi_1)$ (dashed lines correspond to induced edges). Resolving over A creates clause $(B \vee C \vee E)$ which corresponds to a new edge

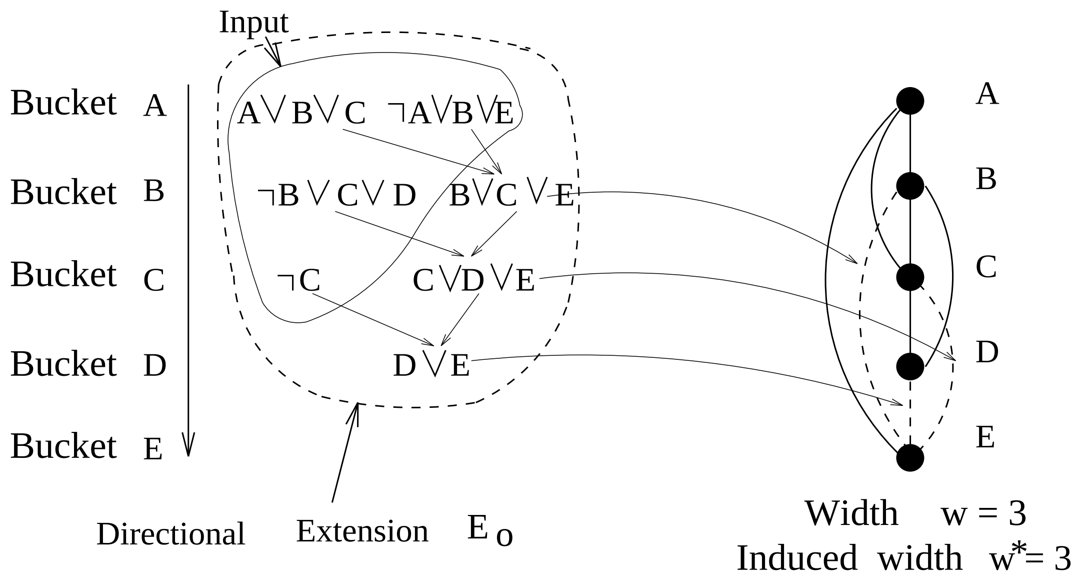


Figure 7: The effect of algorithm DR on the interaction graph of theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$ along the ordering $o = (E, D, C, B, A)$.

between nodes B and E, while resolving over B creates clause $(C \vee D \vee E)$ which induces a new edge between C and E. In general, processing a bucket of a variable Q produces resolvents that connect all the variables mentioned in that bucket. The concepts of induced graph and induced width are defined to reflect those changes.

Definition 1: Given a graph G, and an ordering of its nodes o, the *parent set* of a node X_i is the set of nodes connected to X_i that precede X_i in o. The size of this parent set is called the *width* of X_i relative to o. The *width of the graph* along o, denoted w_o, is the maximum width over all variables. The *induced graph* of G along o, denoted $I_o(G)$, is obtained as follows: going from $i = n$ to $i = 1$, we connect all the neighbors of X_i preceding it in the ordering. The induced width of G along o, denoted w_o^*, is the width of $I_o(G)$ along o, while the induced width w^* of G is the minimum induced width along any ordering.

For example, in Figure 7 the induced graph $I_o(G)$ contains the original (bold) and the induced (dashed) edges. The width of B is 2, while its induced width is 3; the width of C is 1, while its induced width is 2. The maximum width along o is 3 (the width of A), and the maximum induced width is also 3 (the induced width of A and B). Therefore, in this case, the width and the induced width of the graph coincide. In general, however, the induced width of a graph can be significantly larger than its width. Note that in

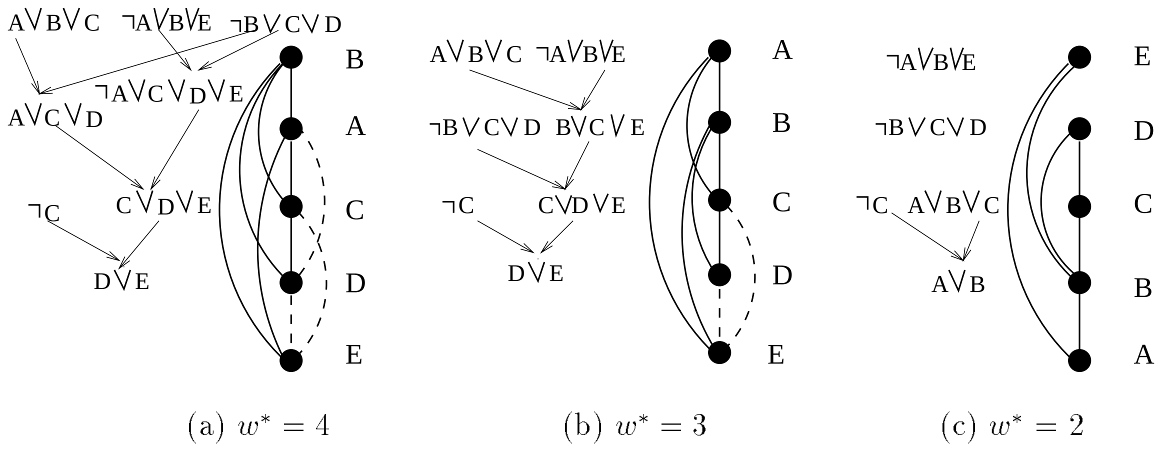


Figure 8: The effect of the ordering on the induced width: interaction graph of theory $\varphi_1 = \{(\neg C), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$ along the orderings (a) $o_1 = (E, D, C, A, B)$, (b) $o_2 = (E, D, C, B, A)$, and (c) $o_3 = (A, B, C, D, E)$.

this example the graph of the directional extension, $G(E_o(\varphi))$, coincides with the induced ordered graph of the input theory's graph, $I_o(G(\varphi))$. Generally,

Lemma 1: *Given a theory φ and an ordering o, $G(E_o(\varphi))$ is a subgraph of $I_o(G(\varphi))$.* $\Box$

The parents of node X_i in the induced graph correspond to the variables mentioned in *bucket*$_i$. Therefore, the induced width of a node can be used to estimate the size of its bucket, as follows:

Lemma 2: *Given a theory φ and an ordering $o = (Q_1, ..., Q_n)$, if Q_i has at most k parents in the induced graph along o, then the bucket of a variable Q_i in $E_o(\varphi)$ contains no more than 3^{k+1} clauses.* $\Box$

We can now derive a bound on the complexity of directional resolution using properties of the problem's interaction graph.

Theorem 4: *(complexity of DR)*
Given a theory φ and an ordering of its variables o, the time complexity of algorithm DR along o is $O(n \cdot 9^{w_o^*})$, and the size of $E_o(\varphi)$ is at most $n \cdot 3^{w_o^*+1}$ clauses, where w_o^* is the induced width of φ's interaction graph along o. $\Box$

Corollary 1: *Theories having bounded w_o^* for some ordering o are tractable.* $\Box$.

Figure 8 demonstrates the effect of variable ordering on the induced width, and consequently, on the complexity of DR when applied to theory φ_1. While DR generates 3 new

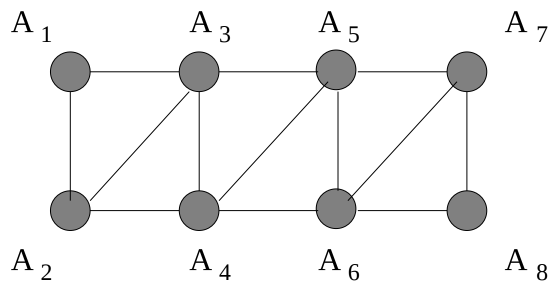


Figure 9: The interaction graph of φ_4 in example 4: $\varphi_4 = \{(A_1 \vee A_2 \vee \neg A_3), (\neg A_2 \vee A_4), (\neg A_2 \vee A_3 \vee \neg A_4), (A_3 \vee A_4 \vee \neg A_5), (\neg A_4 \vee A_6), (\neg A_4 \vee A_5 \vee \neg A_6), (A_5 \vee A_6 \vee \neg A_7), (\neg A_6 \vee A_8), (\neg A_6 \vee A_7 \vee \neg A_8)\}$.

clauses of length 3 along ordering (a), only one binary clause is generated along ordering (c). Although finding an ordering that yields the smallest induced width is NP-hard [1], good heuristic orderings are currently available [6, 14, 55] and continue to be explored [4]. Furthermore, there is a class of graphs, known as *k-trees*, that have $w^* < k$ and can be recognized in $O(n \cdot exp(k))$ time [1].

Definition 2: (k-trees)
1. A clique of size k (complete graph with k nodes) is a k-tree.
2. Given a k-tree defined on $X_1, ..., X_{i-1}$, a k-tree on $X_1, ..., X_i$ can be generated by selecting a clique of size k and connecting X_i to every node in that clique.

Corollary 2: *If the interaction graph of a theory φ having n variables is a subgraph of a k-tree, then there is an ordering o such that the space complexity of algorithm DR along o (the size of $E_o(\varphi)$) is $O(n \cdot 3^k)$, and its time complexity is $O(n \cdot 9^k)$.* $\Box$

Important tractable classes are *trees* ($w^* = 1$) and *series-parallel networks* ($w^* = 2$). These classes can be recognized in polynomial (linear or quadratic) time.

Example 4: Consider a theory φ_n defined on the variables $\{A_1, A_2, ..., A_n\}$. A clause $(A_i \vee A_{i+1} \vee \neg A_{i+2})$ is defined for each odd i, and two clauses $(\neg A_i \vee A_{i+2})$ and $(\neg A_i \vee A_{i+1} \vee \neg A_{i+2})$ are defined for each even i, where $1 \leq i \leq n$. The interaction graph of φ_n for $n = 5$ is shown in Figure 9. The reader can verify that the graph is a 3-tree ($w^* = 2$) and that its induced width along the original ordering is 2. Therefore, by theorem 4, the size of the directional extension will not exceed $27n$.

4.1.1 2-SAT

Note that algorithm DR is tractable for 2-cnf theories, because 2-cnfs are closed under resolution (the resolvents are of size 2 or less) and because the overall number of clauses of size 2 is bounded by $O(n^2)$ (in this case, unordered resolution is also tractable), yielding $O(n \cdot n^2) = O(n^3)$ complexity. Therefore,

Theorem 5: *Given a 2-cnf theory φ, its directional extension $E_o(\varphi)$ along any ordering o is of size $O(n^2)$, and can be generated in $O(n^3)$ time.*

Obviously, DR is not the best algorithm for solving 2-SAT, since 2-SAT can be solved in linear time [26]. Note, however, that DR also compiles the theory into one that can produces each model in linear time. As shown in [17], in this case all models can be generated in output linear time.

4.1.2 The graphical effect of unit resolution

Resolution with a unit clause Q or $\neg Q$ deletes the opposite literal over Q from all relevant clauses. It is equivalent to assigning a value to variable Q. Therefore, unit resolution generates clauses on variables that are already connected in the graph, and therefore will not add new edges.

4.2 Diversity

The concept of induced width sometimes leads to a loose upper bound on the number of clauses recorded by DR. In Example 4, only six clauses were generated by DR, even without eliminating subsumption and tautologies in each bucket, while the computed bound is $27n = 27 \cdot 8 = 216$. Consider the two clauses $(\neg A \vee B)$ and $(\neg C \vee B)$ and the order $o = A, C, B$. When bucket B is processed, no clause is added because B is positive in both clauses, yet nodes A and C are connected in the induced graph. In this subsection, we introduce a new parameter called *diversity*, that provides a tighter bound on the number of resolution operations in the bucket. Diversity is based on the fact that a proposition can be resolved upon only when it appears both positively and negatively in different clauses.

Definition 3: (diversity)
Given a theory φ and an ordering o, let Q_i^+ (Q_i^-) denote the number of times Q_i appears positively (negatively) in $bucket_i$. The *diversity of* Q_i relative to o, $div(Q_i)$, is defined as $Q_i^+ \times Q_i^-$. The *diversity of an ordering* o, $div(o)$, is the largest diversity of its variables relative to o, and the *diversity of a theory*, div, is the minimal diversity among all orderings.

The concept of diversity yields new tractable classes. For example, if o is an ordering having a zero diversity, algorithm DR adds no clauses to φ, regardless of its induced width.

Example 5: Let $\varphi = \{(G \vee E \vee \neg F), (G \vee \neg E \vee D), (\neg A \vee F), (A \vee \neg E), (\neg B \vee C \vee \neg E), (B \vee C \vee D)\}$. It is easy to see that the ordering $o = (A, B, C, D, E, F, G)$ has diversity 0 and induced width 4.

Theorem 6: *Zero-diversity theories are tractable for DR: given a zero-diversity theory φ having n variables and c clauses, 1. its zero-diversity ordering o can be found in $O(n^2 \cdot c)$ time and 2. DR along o takes linear time.* $\Box$

The proof follows immediately from Theorem 8 (see subsection 4.3).

Zero-diversity theories generalize the notion of causal theories defined for general constraint networks of multivalued relations [22]. According to this definition, theories are *causal* if there is an ordering of the propositional variables such that each bucket contains a single clause. Consequently, the ordering has zero diversity. Clearly, when a theory has a non-zero diversity, it is still better to place zero-diversity variables last in the ordering, so that they will be processed first. Indeed, the *pure literal rule* of the original Davis-Putnam resolution algorithm requires processing first all-positive and all-negative (namely, zero-diversity) clauses.

However, the parameter of real interest is the diversity of the directional extension $E_o(\varphi)$, rather than the diversity of φ.

Definition 4: (induced diversity)
The *induced diversity of an ordering o*, $div^*(o)$, is the diversity of $E_o(\varphi)$ along o, and the *induced diversity of a theory*, div^*, is the minimal induced diversity over all its orderings.

Since $div^*(o)$ bounds the number of clauses generated in each bucket, the size of $E_o(\varphi)$ for every o can be bounded by $|\varphi| + n \cdot div^*(o)$. The problem is that computing $div^*(o)$ is generally not polynomial (for a given o), except for some restricted cases. One such case is the class of zero-diversity theories mentioned above, where $div^*(o) = div(o) = 0$. Another case, presented below, is a class of theories having $div^* = 1$. Note that we can easily create examples with high w^* having $div^* \leq 1$.

Theorem 7: *Given a theory φ defined on variables $Q_1, \ldots, Q_n$, such that each symbol Q_i either (a) appears only negatively (only positively), or (b) it appears in exactly two clauses, then $div^*(\varphi) \leq 1$ and φ is tractable.* $\Box$

4.3 Ordering heuristics

As previously noted, finding a minimum-induced-width ordering is known to be NP-hard [1]. A similar result can be demonstrated for minimum-induced-diversity orderings. However, the corresponding suboptimal (non-induced) min-width and min-diversity heuristic

```
min-diversity ($\varphi$)
1. For $i = n$ to 1 do:
        Choose symbol $Q$ having the smallest diversity
        in $\varphi - \bigcup_{j=i+1}^{n} bucket_j$ and put it in the $i^{th}$ position.
```

Figure 10: Algorithm *min-diversity*.

```
min-width ($\varphi$)
1. Initialize: $G \leftarrow G(\varphi)$
2. For $i = n$ to 1 do
        1.1. Choose symbol $Q$ having the smallest
             degree in $G$ and put it in the $i^{th}$ position.
        1.2. $G \leftarrow G - \{Q\}$.
```

Figure 11: Algorithm *min-width*.

orderings often provide relatively low induced width and induced diversity. Min-width and min-diversity orderings can be computed in polynomial time by a simple greedy algorithm, as shown in Figures 10 and 11.

Theorem 8: *Algorithm* min-diversity *generates a minimal diversity ordering of a theory in time $O(n^2 \cdot c)$, where n is the number of variables and c is the number of clauses in the input theory.* $\square$

The *min-width* algorithm [14] (Figure 11) is similar to the min-diversity, except that at each step we select a variable with the smallest *degree* in the current interaction graph. The selected variable is then placed i-th in the ordering and deleted from the graph.

A modification of min-width ordering, called *min-degree* [28] (Figure 12), connects all the neighbors of the selected variable in the current interaction graph before the variable is deleted. Empirical studies demonstrate that the min-degree heuristic usually yields lower-w^* orderings than the induced-width heuristic. In all these heuristics ties are broken randomly.

There are several other commonly used ordering heuristics, such as *max-cardinality* heuristic presented in Figure 13. For more details, see [6, 14, 55].

```
min-degree ($\varphi$)
1. Initialize: $G \leftarrow G(\varphi)$
2. For $i = n$ to 1 do
        1.1. Choose symbol $Q$ having the smallest
             degree in $G$ and put it in the $i^{th}$ position.
        1.2. Connect the neighbors of $Q$ in $G$.
        1.3. $G \leftarrow G - \{Q\}$.
```

Figure 12: Algorithm *min-degree*.

```
max-cardinality ($\varphi$)
1. For $i = 1$ to n do
        Choose symbol $Q$ connected to maximum number of
        previously ordered nodes in $G$ and put it in the $i^{th}$ position.
```

Figure 13: Algorithm *max-cardinality*.

5 Directional Resolution and Tree-Clustering

In this section we further discuss the knowledge-compilation aspects of directional resolution, and relate it to *tree-clustering* [21], a general preprocessing technique commonly used in constraint and belief networks.

As stated in Theorem 2, given an input theory and a variable ordering, algorithm DR produces a directional extension that allows model generation in linear time. Also, when entailment queries are restricted to a small fixed subset of the variables C, orderings initiated by the queried variables are preferred, since in such cases only a subset of the directional extension needs to be processed. The complexity of entailment in this case is $O(exp(min(|C|, w_o^*)))$, when w_o^* is computed over the induced graph truncated above variables in C [3].

However, when queries are expected to be uniformly distributed over all the variables it

[3]Moreover, since querying variables in C implies the addition of unit clauses, all the edges incident to the queried variables can be deleted, further reducing the induced width.

may be worthwhile to generate a compiled theory symmetrical with regard to all variables. This can be accomplished by tree-clustering [21], a compilation scheme used for constraint networks. Since cnf theories are special types of constraint networks, tree-clustering is immediately applicable. The algorithm compiles the propositional theory into a *join-tree* of relations (i.e., partial models) defined over *cliques* of variables that interact in a tree-like manner. The join-tree allows query processing in linear time. A tree-clustering algorithm for propositional theories presented in [5], is described in Figure 14 while a variant of tree-clustering that generates a join-tree of clauses rather than a tree of models is presented later.

```
Tree-clustering ($\varphi$)
Input: A $cnf$ theory $\varphi$ and its interaction graph $G(\varphi)$, an ordering $o$.
Output: A join-tree representation of all models of $\varphi$, $TCM(\varphi)$.
Graph operations:
1. Apply triangulation to $G_o(\varphi)$ yielding a chordal graph $G_h = I_o(G)$.
2. Let $C_1$,...,$C_t$ be all the maximal cliques in $G_h$ indexed by their highest nodes.
3. For each $C_i$, $i = t$ to 1,
   connect $C_i$ to $C_j$ $(j < i)$, where $C_j$ shares the largest set of variables with $C_i$.
The resulting graph is called a join tree T.
4. Assign each clause to every clique that contains all its atoms, yielding $\varphi_i$ for each $C_i$.
Model generation:
5. For each clique $C_i$, compute $M_i$, the set of models over $\varphi_i$.
6. Apply arc-consistency on the join tree $T$ of models:
   for each $C_i$, and for each $C_j$ adjacent to $C_i$ in T, delete from $M_i$ every model $M$
   that does not agree with any model in $M_j$ on the set of their common variables.
6. Return $TCM_o(\varphi) = \{M_1, ..., M_t\}$ and the tree structure.
```

Figure 14: Model-based tree-clustering (TC).

The first three steps of tree-clustering (TC) are applied only to the interaction graph of the theory, transforming it into a *chordal* graph (a graph is chordal if every cycle of length at least four has a chord, i.e. an edge between two non-sequential nodes in that cycle). This procedure, called *triangulation* [61], processes the nodes along some order of the variables o, going from the last node to the first, connecting edges between the earlier neighbors of each node. The result is the induced graph along o, which is chordal, and whose maximal cliques serve as the nodes in the resulting structure called a join-tree. The size of the largest clique in the triangulated (induced) graph equals $w^*_o + 1$. Steps 2 and

3 of the algorithm complete the *join-tree* construction by connecting the various cliques into a tree structure. Once the tree of cliques is identified, each clause in φ is placed in *every* clique that contains its variables (step 4), yielding subtheories φ_i for each clique C_i. In step 5, the models M_i of each φ_i are computed and replace φ_i. Finally (step 6), arc-consistency is enforced on the tree of models (for more details see [21, 5]). Given a theory φ, the algorithm generates a tree of partial models denoted $TCM(\varphi)$.

It was shown that a join-tree yields a tractable representation. Namely, satisfiability, model generation, and a variety of entailment queries can all be done in linear or polynomial time:

Theorem 9: *[5]*

1. *A theory φ and a $TCM(\varphi)$ generated by algorithm TC are satisfiable if and only if none of $M_i \in TCM(\varphi)$ is empty. This can be verified in linear time in the resulting join-tree.*

2. *Deciding whether a literal $P \in C_i$ is consistent with φ can be done in linear time in $|M_i|$, by scanning the columns of a relation M_i defined over P.*

3. *Entailment of a clause α can be determined in $O(|\alpha| \cdot n \cdot m \cdot log(m))$ time, where m bounds the number of models in each clique. This is done by temporary elimination of all submodels that disagree with α from all relevant cliques, and reapplying arc-consistency.* $\Box$

We now present a variant of the tree-clustering algorithm where each clique in the final output join-tree is associated with a subtheory of clauses rather than with a set of models, while all the desirable properties of the compiled representation are maintained. We show that the compiled subtheories are generated by two successive applications of DR along an ordering dictated by the join-tree's structure. The resulting algorithm *Clause-based Tree-Clustering (CTC)* (Figure 15) outputs a clause-based join-tree, denoted $TCC(\varphi)$.

The first three steps of structuring the join-tree and associating each clique with cnf subtheories (step 4) remain unchanged. Directional resolution is then applied to the resulting tree of cliques *twice*, from leaves to the root and vice-versa. However, DR is modified; each bucket is associated with a clique rather than with a single variable. Thus, each clique is processed by *full (unordered)* resolution relative to *all* the variables in the cliques. Some of the generated clauses are then copied into the next neighboring clique. Let $o = C_1 ... \, C_t$ be a *tree-ordering* of cliques generated by either breadth-first or depth-first traversal of the clique-tree rooted at C_1. For each clique, the rooted tree defines a parent and a set of child cliques. Cliques are then processed in a reverse order of o. When

```
CTC($\varphi$)
Input: A $cnf$ theory $\varphi$ and its interaction graph $G(\varphi)$, an ordering $o$.
Output: A clause-based join-tree representation of $\varphi$
1. Compute the skeleton join-tree (steps 1-3 in Figure 14.
2. Place every clause in very clique that contains it literals.
Let $C_1$,...,$C_t$ be a breadth first search ordering of the clique-tree that starts with $C_1$
as its root. Let $\varphi_1$,...,$\varphi_t$ be theories in $C_1$,...,$C_t$, respectively.
3. For $i = t$ to 1, $\varphi_i \leftarrow res(\varphi_i)$ (namely, close $\varphi_i$ under resolution)
put a copy of resolvents defined only on variables shared between $C_i$ and $C_j$ where
$C_j$ is an earlier clique, into $C_j$.
4. For $i = 1$ to t do $C_i \leftarrow res(C_i)$;
put a copy of resolvents defined only on variables that $C_i$ shares with a later clique $C_j$,
into $C_j$.
5. Return $TCC(\varphi) = \{\varphi_1^*, ..., \varphi_t^*\}$, the set of all clauses defined
on each clique and the tree structure.
```

Figure 15: Algorithm clause-based tree-clustering (CTC).

processing clique C_i and its subtheory φ_i, all possible resolvents over the variables in C_i are added to φ_i. The resolvents defined only on variables shared by C_i and its parent C_l are copied and placed into C_l. The second phase works similarly in the opposite direction, from the root C_1 towards the leaves. In this case, the resolvents generated in clique C_i that are defined on variables shared with a child clique C_j are copied into C_j [4]. Since applying full resolution to theories having $|C|$ variables is time and space exponential in $|C|$ we get:

Theorem 10: *The complexity of CTC is time and space $O(n \cdot exp(w^*))$, where w^* is the induced width of the ordered graph used for generating the join-tree structure.* □

Example 6: Consider theory $\varphi_2 = \{(\neg B \vee A)(A \vee \neg C), (\neg B \vee D), (C \vee D \vee E)\}$. Using the order $o = (A, B, C, D, E)$, directional-resolution along o adds no clauses. The join-tree structure relative to this ordering is obtained by selecting the maximal cliques in the ordered induced graph (see Figure 16a). We get $C_3 = EDC$, $C_2 = BCD$, and $C_1 = ABC$. Step 4 places clause $(C \vee D \vee E)$ in clique C_3, clause$(\neg B \vee D)$ in C_2 and clauses $(A \vee \neg C)$ and $(\neg B \vee A)$ in C_1. The resulting set of clauses in each clique after

[4]Note that duplication of resolvents can be avoided using a simple indexing scheme.

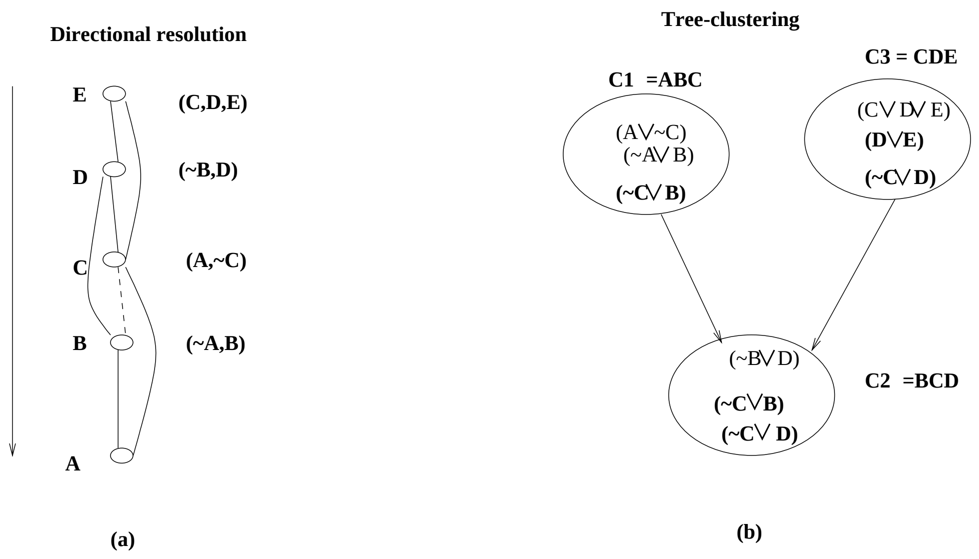


Figure 16: Theory and its two tree-clusterings.

processing by tree-clustering using $o = C_1, C_2, C_3$ is given in Figure 16b. The boldface clauses in each clique are those added during processing. No clause is generated in its backward phase. In the root clique C_1, resolution over A generates clause $(\neg C \vee B)$ which is then added to clique C_2. Processing C_2 generates clause $(\neg C \vee D)$ added to C_3. Finally, processing C_3 generates clause $(D \vee E)$.

The most significant property of the compiled sub-theories of each clique C_i, denoted φ_i^*, is that each contains all the prime implicates of φ defined over variables in C_i. This implies that entailment queries involving only variables contained in a single clique, C_i, can be answered in linear time, scanning the clauses of φ_i^*. Clauses that are *not* contained in one clique can be processed in $O(exp(w^* + 1))$ time.

To prove this claim, we first show that the clause-based join-tree of φ contains the directional extensions of φ along all the orderings that are consistent with the tree-structure. The ability to generate a model backtrack-free facilitated by the directional extensions therefore guarantees the existence of all clique-restricted prime implicates. We provide a formal account of these claims below.

Definition 5: A *prime implicate* of a theory φ is a clause α such that $\varphi \models \alpha$, and there is no $\alpha_1 \subset \alpha$ s.t. $\varphi \models \alpha_1$.

Definition 6: Let φ be a cnf theory, and let C be a subset of the variables of φ. We denote by $prime_\varphi$ the set of all prime implicates of φ, and by $prime_\varphi(C)$ the set of all prime implicates of φ that are defined only on variables in C.

We will show that any compiled clausal tree, $TCC(\varphi)$, contains the directional extension of φ along a variety of variable orderings.

Lemma 3: *Given a theory φ, let $T = TCC(\varphi)$, be a clause-based join-tree of φ and let C be a clique in T. Then, there exist an ordering o that can start with any internal ordering of the variables in C, such that $E_o(\varphi) \subseteq TCC(\varphi)$.* $\Box$

Based on Lemma 3 we can prove the following theorem:

Theorem 11: *Let φ be a theory and let $T = TCC(\varphi)$ be a clause-based join-tree of φ, then for every clique $C \in T$, $prime_\varphi(C) \subseteq TCC(\varphi)$.* $\Box$

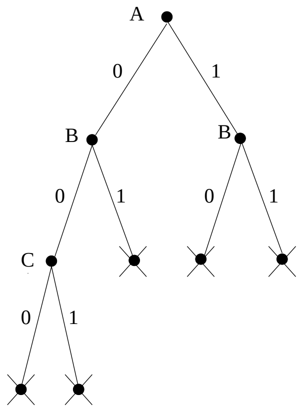
Consider again theory φ_3 and Figure 16. Focusing on clique C_3 we see that it has only two prime implicates, $(D \vee E)$ and$(\neg C \vee D)$.

Having all the prime implicates of a clique has a semantic and a syntactic value. Semantically, it means that all the information related to variables C_i is available inside the compiled theory φ_i^*. The rest of the information is irrelevant. On the syntactic level we also know that φ_i^* is the most explicit representation of this information. From Theorem 11 we conclude:

Corollary 3: *Given a theory φ and its join-tree $TCC_o(\varphi)$, the following properties hold:*

1. *The theory φ is satisfiable if and only if $TCC(\varphi)$ does not contain an empty clause.*
2. *If $T = TCC(\varphi)$ for some φ, then entailment of any clause whose variables are contained in a single clique can be decided in linear time in T.*
3. *Entailment of an arbitrary clause α from φ can be decided in $O(exp(w^* + 1))$ time and space.*
4. *Checking if a new clause is consistent with φ can be done in linear time in T.* $\Box$

In the example shown in Figure 16, the compiled sub-theory associated with clique C_2 is $\varphi_2^* = \{(\neg B \vee D), (\neg C \vee B), (\neg C \vee D)\}$. To determine if φ entails $\alpha = (C \vee B \vee D)$, we must assess whether or not α is contained in φ_2^*. Since it is not contained, we conclude that it is not entailed. To determine if α is consistent with φ, we must see if φ entails the negation of each literal. If it does, the clause is inconsistent. Since φ_2^* does not include $\neg B$, $\neg C$, or $\neg D$, neither of those literals is entailed by φ, and therefore, α is consistent with φ.



(a)

```
DP(φ):
Input: A cnf theory φ.
Output: A decision of whether φ is satisfiable.
1. Unit_propagate(φ);
2. If the empty clause generated return(false);
3. else if all variables are assigned return(true);
4. else
5.       Q = some unassigned variable;
6.       return(DP(φ ∧ ¬Q) ∨
7.              DP(φ ∧ Q) )
```

(b)

Figure 17: (a) A backtracking search tree along the ordering A, B, C for a cnf theory $\varphi_5 = \{(\neg A \vee B), (\neg C \vee A), \neg B, C\}$ and (b) the Davis-Putnam Procedure.

6 Backtracking Search (DP)

Backtracking search processes the variables in some order, instantiating the next variable if it has a value consistent with previous assignments. If there is no such value (a situation called a *dead-end*), the algorithm *backtracks* to the previous variable and selects an alternative assignment. Should no consistent assignment be found, the algorithm backtracks again. The algorithm explores the *search tree*, in a depth-first manner, until it either finds a solution or concludes that no solution exists. An example of a search tree is shown in Figure 17a. This tree is traversed when deciding satisfiability of a propositional theory $\varphi_5 = \{(\neg A \vee B), (\neg C \vee A), \neg B, C\}$. The tree nodes correspond to the variables, while the tree branches correspond to different assignments (0 and 1). Dead-end nodes are crossed out. Theory φ_5 is obviously inconsistent.

There are various advanced backtracking algorithms for solving CSPs that improve the basic scheme using "smart" variable- and value-ordering heuristics ([9], [33]). More efficient backtracking mechanisms, such as *backjumping* [36, 13, 50], constraint propagation (e.g., *arc-consistency*, *forward checking* [41]), or learning (recording constraints) [13, 31, 2] are available. The Davis-Putnam Procedure (DP) [11] shown in Figure 17b is a backtracking search algorithm for deciding propositional satisfiability combined with unit propagation. Various branching heuristics augmenting this basic version of DP have been proposed since 1962 [44, 9, 42, 38].

The worst-case time complexity of all backtracking algorithms is exponential in the

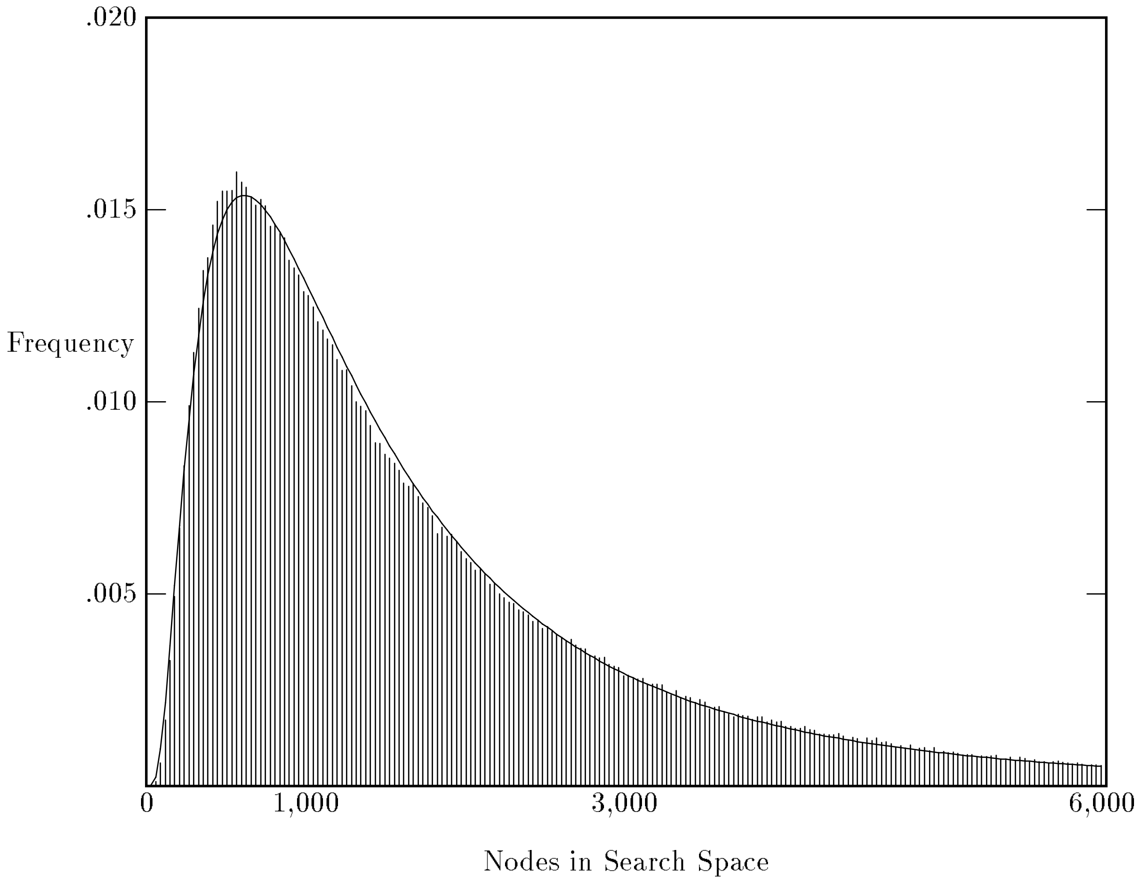


Figure 18: An empirical distribution of the number of nodes explored by algorithm BJ-DVO (backjumping+dynamic variable ordering) on 10^6 instances of inconsistent random binary CSPs having N=50 variables, domain size D=6, constraint density C=.1576 (probability of a constraint between two variables), and tightness T=0.333 (the fraction of prohibited value pairs in a constraint).

number of variables while their space complexity is linear. Yet, the average time complexity of DP depends on the distribution of instances [29] and is often much lower then its worst-case bound. Usually, its average performance is affected by rare, but exceptionally hard instances. Exponential-family empirical distributions (e.g., lognormal, Weibull) proposed in recent studies [32, 54] summarize such observations in a concise way. A typical distribution of the number of explored search-tree nodes is shown in Figure 18. The distribution is shown for inconsistent problems. As it turns out, consistent and inconsistent CSPs produce different types of distributions (for more details see [32, 33]).

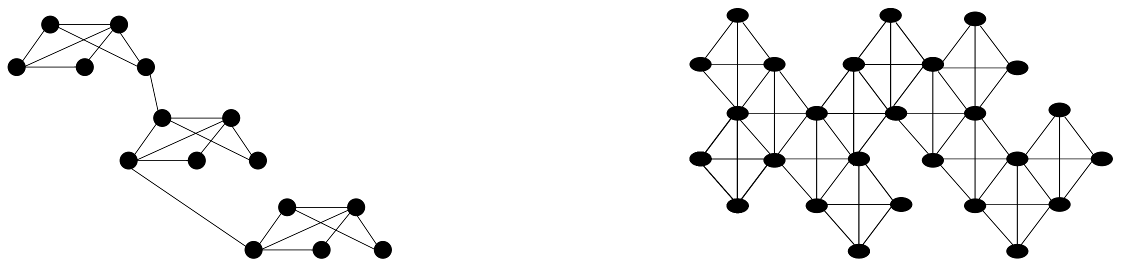


Figure 19: An example of a theory with (a) a chain structure (3 subtheories, 5 variables in each) and (b) a (k,m)-tree structure (k=2, m=2).

7 DP versus DR: Empirical Evaluation

In this section we present an empirical comparison of DP and DR on different types of cnf theories, including uniform random problems, random chains and (k,m)-trees, and benchmark problems from the Second DIMACS Challenge [5]. The algorithms were implemented in C and tested on SUN Sparc stations. Since we used several machines having different performance (from Sun 4/20 to Sparc Ultra-2), we specify which machine was used for each set of experiments. Reported runtime is measured in seconds.

Algorithm DR is implemented as discussed in Section 3. If it is followed by DP using the same fixed variable ordering, no dead-ends will occur (see Theorem 2).

Algorithm DP was implemented using the dynamic variable ordering heuristic of *Tableau* [9], a state-of-the-art backtracking algorithm for SAT. This heuristic, called the *2-literal-clause* heuristic, suggests instantiating next a variable that would cause the largest number of unit propagations approximated by the number of 2-literal clauses in which the variable appears. The augmented algorithm significantly outperforms DP without this heuristic [9].

7.1 Random problem generators

To test the algorithms on problems with different structures, several random problem generators were used. The *uniform k-cnfs* generator [49] uses as input the number of variables N, the number of clauses C, and the number of literals per clause k. Each clause is generated by randomly choosing k out of N variables and by determining the sign of each literal (positive or negative) with probability p. In the majority of our experiments $p = 0.5$. Although we did not check for clause uniqueness, for large N it is unlikely that identical clauses will be generated.

[5]Available at *ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/volume/cnf.*

Our second generator, *chains*, creates a sequence of independent uniform k-cnf theories (called *subtheories*) and connects each pair of successive cliques by a 2-cnf clause containing variables from two consecutive subtheories in the chain (see Figure 19a). The generator parameters are the number of cliques, $Ncliq$, the number of variables per clique, N, and the number of clauses per clique, C. A chain of cliques, each of size N variables, is a subgraph of a k-tree [1] where $k = 2n - 1$ and therefore, has $w^* \leq 2n - 1$.

We also used a *(k,m)-tree* generator which generates a tree of cliques each having $(k + m)$ nodes where k is the size of the intersection between two neighboring cliques (see Figure 19b, where $k = 2$ and $m = 2$). Given k, m, the number of cliques $Ncliq$, and the number of clauses per clique $Ncls$, the *(k,m)-tree* generator produces a clique of size $k + m$ with $Ncls$ clauses and then generates each of the other $Ncliq - 1$ cliques by selecting randomly an existing clique and its k variables, adding m new variables, and generating $Ncls$ clauses on that new clique. Since a k-m-tree can be embedded into a $(k + m - 1)$-tree, its induced width is bounded by $k + m - 1$ (note that $(k, 1)$-trees are conventional k-trees).

7.2 Results

As expected, on uniform random 3-cnfs having large w^*, the complexity of DR grew exponentially with the problem density while the performance of DP was much better. Even small problems having 20 variables already demonstrate the exponential behavior of DR (see Figure 20a). On larger problems DR often ran out of memory. We did not proceed with more extensive experiments in this case, since the exponential behavior of DR on uniform 3-cnfs is already well-known [35, 39].

However, the behavior of the algorithms on chain problems was completely different. DR was by far more efficient than DP, as can be seen from Table 1 and from Figure 20b, summarizing the results on 3-cnf chain problems that contain 25 subtheories, each having 5 variables and 9 to 23 clauses (24 additional 2-cnf clauses connect the subtheories in the chain) [6]. A min-diversity ordering was used for each instance. Since the induced width of these problems was small (less than 6, on average), directional resolution solved these problems quite easily. However, DP-backtracking encountered rare but extremely hard problems that contributed to its average complexity. Table 2 lists the results on selected hard instances from Table 1 (where the number of dead-ends exceeds 5,000).

Similar results were obtained for other chain problems and with different variable orderings. For example, Figure 21 graphs the experiments with min-width and input orderings. We observe that min-width ordering may significantly improve the performance

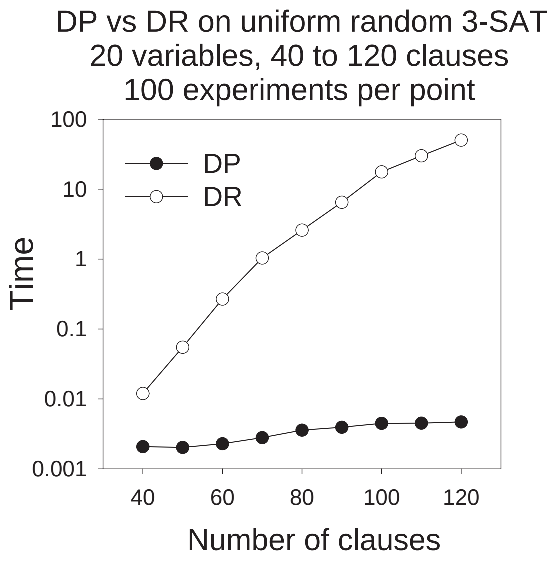
[6]Figure 20b also shows the results for algorithms BDR-DP and backjumping discussed later.

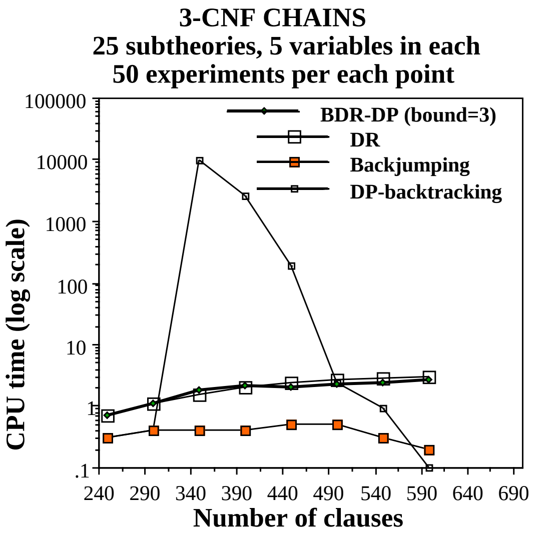
Table 1: DR versus DP on 3-cnf chains having 25 subtheories, 5 variables in each, and from 11 to 21 clauses per subtheory (total 125 variables and 299 to 549 clauses). 20 instances per row. The columns show the percentage of satisfiable instances, time and deadends for DP, time and the number of new clauses for DR, the size of largest clause, and the induced width w^*_{md} along the min-diversity ordering. The experiments were performed on Sun 4/20 workstation.

Num of cls	% sat	**DP** Time	**DP** Dead ends	**DR** Time	**DR** Number of new clauses	**DR** Size of max clause	**DR** w^*
299	100	0.4	1	1.4	105	4.1	5.3
349	70	9945.7	908861	2.2	131	4.0	5.3
399	25	2551.1	207896	2.8	131	4.0	5.3
449	15	185.2	13248	3.7	135	4.0	5.5
499	0	2.4	160	3.8	116	3.9	5.4
549	0	0.9	9	4.0	99	3.9	5.2

Table 2: DR and DP on hard chains when the number of dead-ends is larger than 5,000. Each chain has 25 subtheories, with 5 variables in each (total of 125 variables). The experiments were performed on Sun 4/20 workstation.

Num of cls	Sat: 0 or 1	**DP** Time	**DP** Dead ends	**DR** Time
349	0	41163.8	3779913	1.5
349	0	102615.3	9285160	2.4
349	0	55058.5	5105541	1.9
399	0	74.8	6053	3.6
399	0	87.7	7433	3.1
399	0	149.3	12301	3.1
399	0	37903.3	3079997	3.0
399	0	11877.6	975170	2.2
399	0	841.8	70057	2.9
449	1	655.5	47113	5.2
449	0	2549.2	181504	3.0
449	0	289.7	21246	3.5





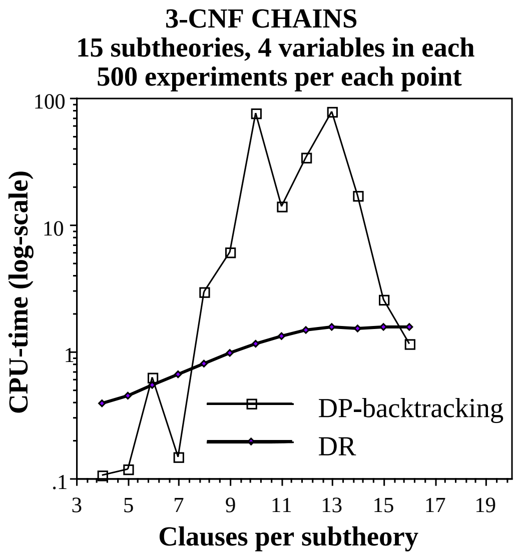
(a) uniform random 3-cnfs, $w^* = 10$ to 18 (b) chain 3-cnfs, $w^* = 4$ to 7

Figure 20: (a) DP versus DR on uniform random 3-cnfs; (b) DP, DR, BDR-DP(3) and backjumping on 3-cnf chains (Sun 4/20).

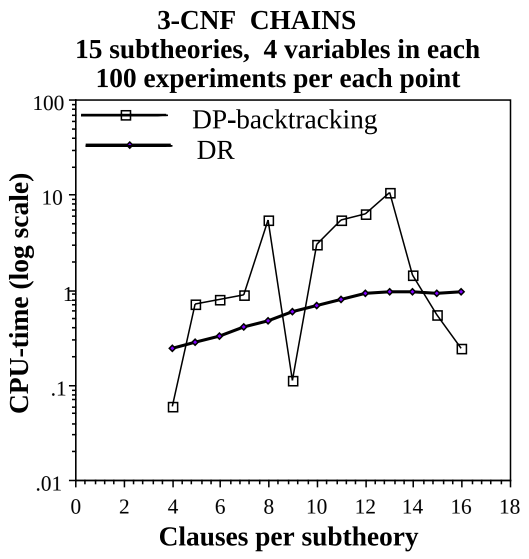
of DP relative to the input ordering (compare Figure 21a and Figure 21b). Still, it did not prevent backtracking from encountering rare, but extremely hard instances.

Table 3 presents the histograms demonstrating the performancs of DP on chains in more details. The histograms show that in most cases the frequency of easy problems (e.g., less than 10 deadends) decreased and the frequency of hard problems (e.g., more than 10^4 deadends) increased with increasing number of cliques and with increasing number of clauses per clique. Further empirical studies are required to investigate the possible *phase transition* phenomenon in chains as it was done for uniform random 3cnfs [7, 49, 9].

In our experiments nearly all of the 3-cnf chain problems that were difficult for DP were unsatisfiable. One plausible explanation is that inconsistent chain theories may have an unsatisfiable subtheory only at the end of the ordering. If all other subtheories are satisfiable then DP will try to re-instantiate variables from the satisfiable subtheories whenever it encounters a dead-end. Figure 22 shows an example of a chain of satisfiable theories with an unsatisfiable theory close to the end of the ordering. Min-diversity and min-width orderings do not preclude such a situation. There are enhanced backtracking schemes, such as *backjumping* [36, 37, 13, 51], that are capable of exploiting the structure and preventing useless re-instantiations. Experiments with backjumping confirm that it



(a) input ordering



(b) min-width ordering

Figure 21: DR and DP on 3-cnf chains with different orderings (Sun 4/20).

Table 3: Histograms of the number of deadends (log-scale) for DP on chains having 20, 25 and 30 subtheories, each defined on 5 variables and 12 to 16 clauses. Each column presents results for 200 instances; each row defines a range of deadednds; each entry is the frequency of instances (out of total 200) that yield the range of deadends. The experiments were performed on Sun Ultra-2.

Deadends	C=12				C=14				C=16		
	Ncliq				Ncliq				Ncliq		
	20	25	30		20	25	30		20	25	30
$[0, 1)$	103	90	75		75	23	8		7	2	2
$[1, 10)$	81	85	102		102	107	93		73	68	59
$[10, 10^2)$	3	4	7		7	21	24		40	37	43
$[10^2, 10^3)$	2	1	4		4	8	12		20	26	22
$[10^3, 10^4)$	1	3	2		2	10	8		21	10	21
$[10^4, \infty)$	10	17	10		10	31	55		39	57	53

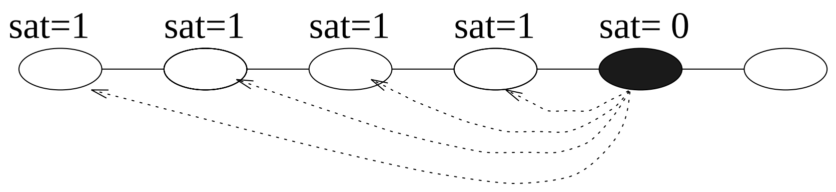


Figure 22: An inconsistent chain problem: a naive backtracking is very inefficient when encountering an inconsistent subproblem at the end of the variable ordering.

Table 4: DP versus Tableau on 150- and 200-variable uniform random 3-cnfs using the min-degree ordering. 100 instances per row. Experiments ran on Sun Sparc Ultra-2.

Cls	% sat	Tableau time	DP time	DP de
150 variables				
550	1.00	0.3	0.4	81
600	0.93	2.0	3.9	992
650	0.28	4.1	10.1	2439
700	0.04	2.7	7.1	1631
200 variables				
780	0.99	11.6	10.0	1836
820	0.95	48.5	43.7	7742
860	0.40	81.7	125.8	22729
900	0.07	26.6	92.4	17111

substantially outperforms DP on the same chain instances (see Figure 20b).

The behavior of DP and DR on *(k-m)*-trees is similar to that on chains and will be discussed later in the context of hybrid algorithms.

7.2.1 Comparing different DP implementations

One may raise the question whether our (not highly optimized) DP implementation is efficient enough to be representative of backtracking-based SAT algorithms. We answer this question by comparing our DP with the executable code of Tableau [9].

The results for 150- and 200-variable uniform random 3-cnf problems are presented in Table 4. We used min-degree as an initial ordering consulted by both (dynamic-ordering) algorithms Tableau and DP in tie-breaking situations. In most cases, Tableau was 2-4 times faster than DP, while in some DP was faster or comparable to Tableau.

On chains, the behavior pattern of Tableau was similar to that of DP. Table 5 compares the runtime histograms for DP and Tableau on chain problems showing that both

Table 5: Histograms of DP and Tableau runtimes (log-scale) on chains having $Ncliq = 15$, $N = 8$, and C from 21 to 27, 200 instances per column. Each row defines a runtime range, and each entry is the frequency of instances within the range. The experiments were performed on Sun Ultra-2.

Time	C=21	C=23	C=25
Tableau runtime histogram			
$[0, 1)$	195	189	166
$[1, 10)$	0	2	12
$[10, 10^2)$	0	3	14
$[10^2, \infty)$	5	6	8
DP runtime histogram			
$[0, 1)$	193	180	150
$[1, 10)$	2	3	8
$[10, 10^2)$	2	2	11
$[10^2, \infty)$	3	15	31

algorithms were encountering rare hard problems, although Tableau usually encountered hard problems less frequently than DP. Some problem instances that were hard for DP were easy for Tableau, and vice versa.

Thus, although Tableau is often more efficient than our implementation, this difference does not change the key distinctions made between backtracking- and resolution-based approaches. Most of experiments in this paper use our implementation of DP [7].

8 Combining search and resolution

The complementary properties of DP and DR suggest combining both into a hybrid scheme (note that algorithm DP already includes a limited amount of resolution in the form of unit propagation). We will present two general parameterized schemes integrating bounded resolution with search. The hybrid scheme BDR-DP(i) performs bounded resolution prior to search, while the other scheme called DCDR(b) uses it dynamically during search.

```
Bounded Directional Resolution: BDR($i$)
Input: A $cnf$ theory $\varphi$, $o = Q_1, ..., Q_n$, and bound $i$.
Output: The decision of whether $\varphi$ is satisfiable.
If it is, a bounded directional extension $E^i_o(\varphi)$.
1. Initialize: generate a partition of 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:
        resolve each pair $\{(\alpha \lor Q_i), (\beta \lor \neg Q_i)\} \subseteq bucket_i$.
        If $\gamma = \alpha \lor \beta$ is empty, return "$\varphi$ is unsatisfiable"
        else if $\gamma$ contains no more than $i$ propositions,
        add $\gamma$ to the bucket of its highest variable.
3.Return $E^i_o(\varphi) = \bigcup_i bucket_i$.
```

Figure 23: Algorithm Bounded Directional Resolution (BDR).

8.1 Algorithm BDR-DP(i)

The resolution operation helps detecting inconsistent subproblems and thus can prevent DP from unnecessary backtracking. Yet, resolution can be costly. One way of limiting the complexity of resolution is to bound the size of the recorded resolvents. This yields the incomplete algorithm *bounded directional resolution*, or $BDR(i)$, presented in Figure 8.1, where i bounds the number of variables in a resolvent. The algorithm coincides with DR except that resolvents with more than i variables are not recorded. This bounds the size of the directional extension $E^i_o(\varphi)$ and, therefore, the complexity of the algorithm. The time and space complexity of BDR(i) is $O(n \cdot exp(i))$. The algorithm is sound but incomplete. Algorithm $BDR(i)$ followed by DP is named *BDR-DP(i)* [8]. Clearly, BDR-DP(0) coincides with DP while for $i > w^*_o$ BDR-DP(i) coincides with DR (each resolvent is recorded).

8.2 Empirical evaluation of BDR-DP(i)

We tested BDR-DP(i) for different values of i on uniform 3-cnfs, chains, (k,m)-trees, and on DIMACS benchmarks. In most cases, BDR-DP(i) achieved its optimal performance

[7]Having the source code for DP allowed us more control over the experiments (e.g., bounding the number of deadends) than having only the executable code for Tableau.

[8]Note that DP always uses the 2-literal-clauses dynamic variable ordering heuristic.

Table 6: DP versus BDR-DP(i) for $2 \leq i \leq 4$ on uniform random 3-cnfs with 150 variables, 600 to 725 clauses, and positive literal probability $p = 0.5$. The induced width w_o^* along the min-width ordering varies from 107 to 122. Each row presents average values on 100 instances (Sun Sparc 4).

Num	**DP**		**BDR-DP(2)**				**BDR-DP(3)**				**BDR-DP(4)**				w_o^*
of cls	Time	Dead ends	BDR time	DP time	Dead ends	New cls	BDR time	DP time	Dead ends	New cls	BDR time	DP time	Dead ends	New cls	
600	4.6	784	0	4.6	786	0	0.1	4.1	692	16	1.7	8.5	638	731	113
625	8.9	1487	0	8.9	1503	0	0.1	8.2	1346	18	1.9	16.8	1188	805	114
650	11.2	1822	0.1	11.2	1821	0	0.1	10.3	1646	19	2.3	21.4	1421	889	115
675	10.2	1609	0.1	9.9	1570	0	0.1	9.1	1405	21	2.6	19.7	1232	975	116
700	7.9	1214	0.1	7.9	1210	0	0.1	7.5	1116	23	3	16.6	969	1071	117
725	6.1	910	0.1	6.1	904	0	0.1	5.7	820	25	3.5	13.3	728	1169	118

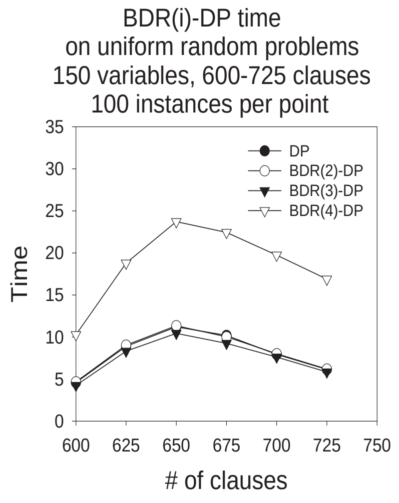
for intermediate values of i.

8.2.1 Performance on uniform 3-cnfs

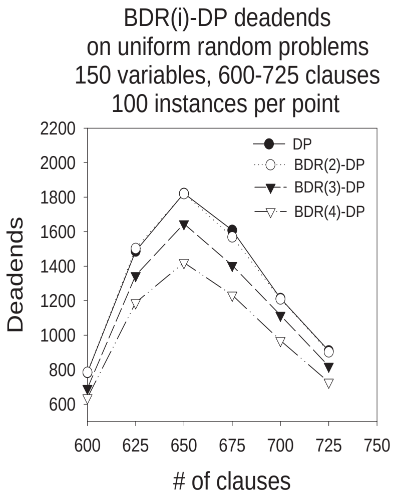
The results for BDR-DP(i) ($0 \leq i \leq 4$) on a class of uniform random 3-cnfs are presented in Table 6. It shows the average time and number of deadends for DP, the average BDR(i) time, DP time and the number of deadends after preprocessing, as well as the average number of new clauses added by BDR(i). An alternative summary of the same data is given in Figure 24, comparing DP and BDR-DP(i) time. It also demonstrates the increase in the number of clauses and the corresponding reduction in the number of deadends. For $i = 2$, almost no new clauses are generated (Figure 24c). Indeed, the graphs for DP and BDR-DP(2) practically coincide. Incrementing i by 1 results in a two orders of magnitude increase in the number of generated clauses, while the number of deadends decreases by 100-200, as shown in Figure 24c.

The results suggest that BDR-DP(3) is the most cost-effective on these problem clases (see Figure 24a). It is slightly faster than DP and BDR-DP(2) (BDR-DP(2) coincides with DP on this problem set) and significantly faster than BDR-DP(4). Table 6 shows that BDR(3) takes only 0.1 second to run, while BDR(4) takes up to 3.5 seconds and indeed generates many more clauses. Observe also that DP runs slightly faster when applied after BDR(3). Interestingly enough, for $i = 4$ the time of DP almost doubles although fewer deadends are encountered. For example, in Table 6, for the problem set with 650 clauses, DP takes on average 11.2 seconds but after preprocessing by BDR(4) it takes 21.4 seconds. This can be explained by the significant increase in the number of clauses that need to be consulted by DP. Thus, as i increases beyond 3, DP's performance is likely to worsen while at the same time the complexity of preprocessing grows exponentially in i. Table 7 presents additional results for problems having 200 variables where $p = 0.7$ [9].

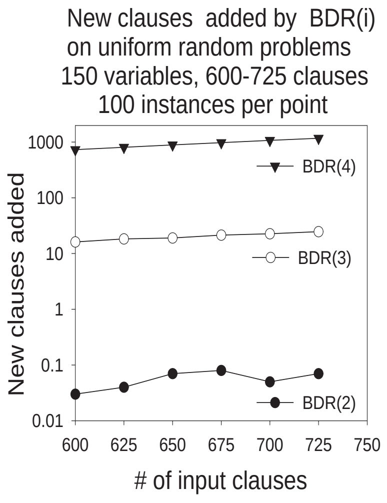
[9]Note that the average decrease in the number of deadends is not always monotonic: for problems



(a) time



(b) deadends



(c) new clauses

Figure 24: BDR-DP(i) on a class of uniform random 3-cnf problems. (150 variables, 600 to 725 clauses). The induced width along the min-width ordering varies from 107 to 122. Each data point corresponds to 100 instances. Note that the plots for DP and BDR(2)-DP in (a) and (b) almost coincide (the white-circle plot for BDR(2)-DP overlaps with the black-circle plot for DP).

Table 7: DP versus BDR-DP(i) for $i = 3$ and $i = 4$ on uniform 3-cnfs with 200 variables, 900 to 1400 clauses, and with positive literal probability $p = 0.7$. Each row presents mean values on 20 experiments.

Num	**DP**		**BDR-DP(3)**				**BDR-DP(4)**			
of cls	Time	Dead ends	BDR time	DP time	Dead ends	New cls	BDR time	DP time	Dead ends	New cls
900	1.1	0	0.3	1.1	0	11	8.4	1.7	1	657
1000	2.7	48	0.4	1.6	14	12	13.1	2.7	21	888
1100	8.8	199	0.6	27.7	685	18	20.0	50.4	729	1184
1200	160.2	3688	0.8	141.5	3271	23	28.6	225.7	2711	1512
1300	235.3	5027	1.0	219.1	4682	28	39.7	374.4	4000	1895
1400	155.0	3040	1.2	142.9	2783	34	54.4	259.0	2330	2332

Table 8: DP versus BDR-DP(3) on uniform random 3-cnfs with $p = 0.5$ at the phase-transition point (C/N=4.3): 150 variables and 645 clauses, 200 variables and 860 clauses, 250 variables and 1075 clauses. The induced width w_o^* was computed for the min-width ordering. The results in the first two rows summarize 100 experiments, while the last row represents 40 experiments.

$< vars, cls >$	**DP**		**BDR-DP(3)**				w_o^*
	Time	Dead ends	BDR time	DP time	Dead ends	New cls	
$< 150, 650 >$	11.2	1822	0.1	10.3	1646	19	115
$< 200, 860 >$	81.3	15784	0.1	72.9	14225	18	190
$< 250, 1075 >$	750	115181	0.1	668.8	102445	19	1094

Finally, we observe that effect of BDR(3) is more pronounced on larger theories. In Table 8 we compare the results for three classes of uniform 3-cnf problems in the phase transition region. While this improvement was marginal for 150-variable problems (from 11.2 seconds for DP to 10.3 seconds for BDR-DP(3)), it was more pronounced on 200-variable problems (from 81.3 to 72.9 seconds), and on 250-variable problems (from 929.9 to 830.5 seconds). In all those cases the average speed-up is about 10%.

Our tentative empirical conclusion is that $i = 3$ is the optimal parameter for BDR-DP(i) on uniform random 3-cnfs.

8.2.2 Performance on chains and (k,m)-trees

The experiments with chains showed that BDR-DP(3) easily solved almost all instances that were hard for DP. In fact, the performance of BDR-DP(3) on chains was comparable to that of DR and backjumping (see Figure 20b).

Experimenting with (k, m)-trees, while varying the number of clauses per clique, we discovered again exceptionally hard problems for DP. The results on (1,4)-trees and on (2,4)-trees are presented in Table 9. In these experiments we terminated DP once it exceeded 20,000 dead-ends (around 700 seconds). This happened in 40% of (1,4)-trees with $Ncls = 13$, and in 20% of (2,4)-trees with $Ncls = 12$. Figure 25 shows a scatter diagram comparing DP and BDR-DP(3) time on the same data set together with an additional 100 experiments on (k,m)-trees having 15 cliques (total of 500 instances).

As in the case of 3-cnf chains we observed that the majority of the exceptionally hard problems were unsatisfiable. For fixed m, when k is small and the number of cliques is

having 1000 clauses, DP has an average of 48 deadends, BDR-DP(3) yields 14 deadends, but BDR-DP(4) yields 21 deadends. This may occur because DP uses dynamic variable ordering.

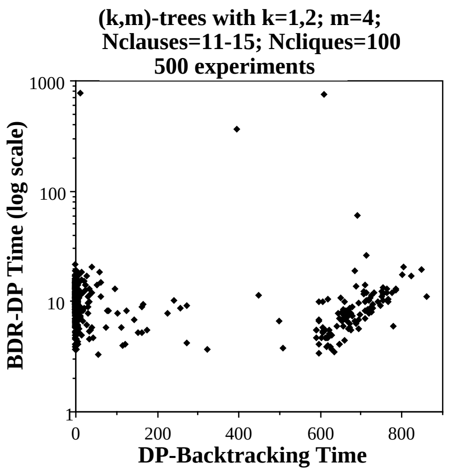
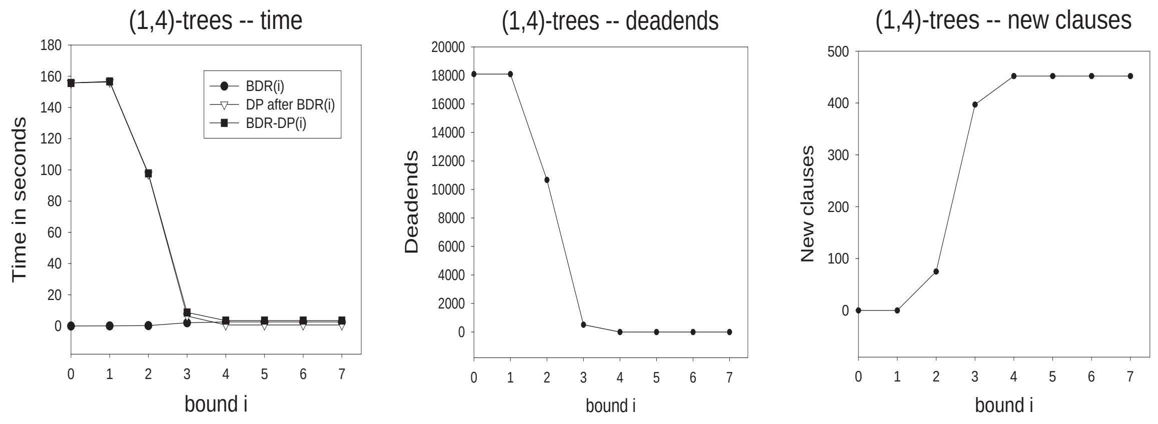


Figure 25: DP and BDR-DP(3) on (k-m)-trees, k=1,2, m=4, Ncliq=100, and Ncls=11 to 15. 50 instances per each set of parameters (total of 500 instances), an instance per point.

Table 9: BDR-DP(3) and DP (termination at 20,000 dead ends) on (k,m)-trees, k=1,2, m=4, Ncliq=100, and Ncls=11 to 14. 50 experiments per each row.

Number of	% sat	**DP**		**BDR-DP(3)**			
		Time	Dead ends	BDR(3) time	DP after BDR(3)		Number of new clauses
					time	dead ends	
(1,4)-tree, *Ncls* = 11 to 14, *Ncliq* = 100 (total: 401 vars, 1100-1400 cls)							
1100	60	233.2	7475	5.4	17.7	2	298
1200	18	352.5	10547	7.5	1.2	7	316
1300	2	328.8	9182	9.8	0.25	3	339
1400	0	174.2	4551	11.9	0.0	0	329
(2,4)-tree, *Ncls* = 11 to 14, *Ncliq* = 100 (total: 402 vars, 1100-1400 cls)							
1100	36	193.7	6111	4.1	23.8	568	290
1200	12	160.0	4633	6.0	1.6	25	341
1300	2	95.1	2589	8.4	0.1	0	390
1400	0	20.1	505	10.3	0.0	0	403



(a) time (b) deadends (c) new clauses

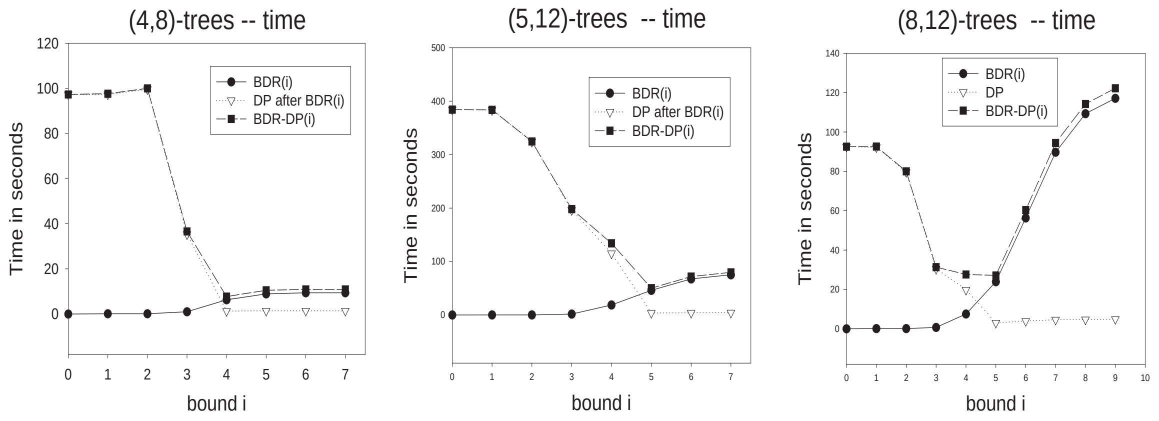
Figure 26: BDR-DP(i) on 100 instances of (1,4)-trees, $Ncliq = 100$, $Ncls = 11$, $w^*_{md} = 4$ (termination at 50,000 deadends). (a) Average time, (b) the number of dead-ends, and (c) the number of new clauses are plotted as functions of the parameter i. Note that the plot for BDR-DP(i) practically coincides with the plot for DP when $i \leq 3$, and with DP when $i > 3$.

large, hard instances for DP appeared more frequently.

The behavior of BDR-DP(i) as a function of i on structured bounded-w^* theories is demonstrated in Figures 26 and 27. In these experiments we used min-degree ordering that yielded smaller average w^* (denoted w^*_{md}) than input ordering, min-width ordering, and min-cardinality ordering (see [52] for details). Figure 26 shows results for (1,4)-trees, while Figure 27a presents the results for (4,8)-trees, (5,12)-trees, and (8,12)-trees. Each point represents an average over 100 instances. We observed that for relatively low-w^* (1,4)-trees preprocessing time is not increasing when $i > 3$ since BDR(4) coincides with DR (Figure 26a), while for high-w^* (8,12)-trees the preprocessing time grows quickly with increasing i (Figure 26c). Since DP time after BDR(i) usually decreases monotonically with i, the total time of BDR-DP(i) is optimal for some intermediate values of i. We observe that for (1,4)-trees, BDR-DP(3) is most efficient, while for (4,8)-trees and for (5,12)-trees the optimal parameters are $i = 4$ and $i = 5$, respectively. For (8,12)-trees, the values $i = 3, 4$, and 5 provide the best performance.

8.2.3 BDR-DP(i), DP, DR, and Tableau on DIMACS benchmarks

We tested DP, Tableau, DR and BDR-DP(i) for i=3 and i=4 on the benchmark problems from the Second DIMACS Challenge. The results presented in Table 10 are quite



(a) (4,8)-trees, $w^*_{md} = 9$ (b) (5,12)-trees, $w^*_{md} = 12$ (c) (8,12)-trees, $w^*_{md} = 14$

Figure 27: BDR-DP(i) on 3 classes of (k,m)-tree problems: (a) (4,8)-trees, $Ncliq = 60$, $Ncls = 23$, $w^*_{md} = 9$, (b) (5,12)-trees, $Ncliq = 60$, $Ncls = 36$, $w^* = 12$, and (c) (8,12)-trees, $Ncliq = 50$, $Ncls = 34$, $w^* = 14$ (termination at 50,000 deadends). 100 instances per each problem class. Average time, the number of dead-ends, and the number of new clauses are plotted as functions of the parameter i.

interesting: while all benchmark problems were relatively hard for both DP and Tableau, some of them had very low w^* and were solved by DR in less than a second (e.g., *dubois20* and *dubois21*). On the other hand, problems having high induced width, such as *aim-100-2_0-no-1* ($w^* = 54$) and *bf0432-007* ($w^* = 131$) were intractable for DR, as expected. Algorithm BDR-DP(i) was often better than both "pure" DP and DR. For example, solving the benchmark *aim-100-2_0-no-1* took more than 2000 seconds for Tableau, more than 8000 seconds for DP, and DR ran out of memory, while BDR-DP(3) took only 0.9 seconds and reduced the number of DP deadends from more than 10^8 to 5. Moreover, preprocessing by BDR(4), which took only 0.6 seconds, made the problem backtrack-free. Note that the induced width of this problem is relatively high ($w^* = 54$).

Interestingly, for some DIMACS problems (e.g., *ssa0432-003* and *bf0432-007*) preprocessing by BDR(3) actually worsened the performance of DP. Similar phenomenon was observed in some rare cases for (k,m)-trees (Figure 25). Still, *BDR-DP(i)* with intermediate values of i is overall more cost-effective than both DP and DR. On unstructured random uniform 3-cnfs BDR-DP(3) is comparable to DP, on low-w^* chains it is comparable to DR, and on intermediate-w^* (k,m)-trees, BDR-DP(i) for $i = 3, 4, 5$ outperforms

Table 10: Tableau, DP, DR, and BDR-DP(i) for i=3 and 4 on the Second DIMACS Challenge benchmarks. The experiments were performed on Sun Sparc 5 workstation.

Problem	**Tableau**	**DP**	Dead	**DR**	**BDR-DP(3)**			**BDR-DP(4)**			w^*
	time	time	ends	time	time	Dead ends	New cls	time	Dead ends	New cls	
aim-100-2_0-no-1	2148	>8988	$> 10^8$	*	0.9	5	26	0.60	0	721	54
dubois20	270	3589	3145727	0.2	349	262143	30	0.2	0	360	4
dubois21	559	7531	6291455	0.2	1379	1048575	20	0.2	0	390	4
ssa0432-003	12	45	4787	4	132	8749	950	40	1902	1551	19
bf0432-007	489	8688	454365	*	46370	677083	10084	*	*	*	131

both DR and DP. We believe that the transition from i=3 to i=4 on uniform problems is too sharp, and that intermediate levels of preprocessing may provide a more refined trade-off.

8.3 Algorithm DCDR(b)

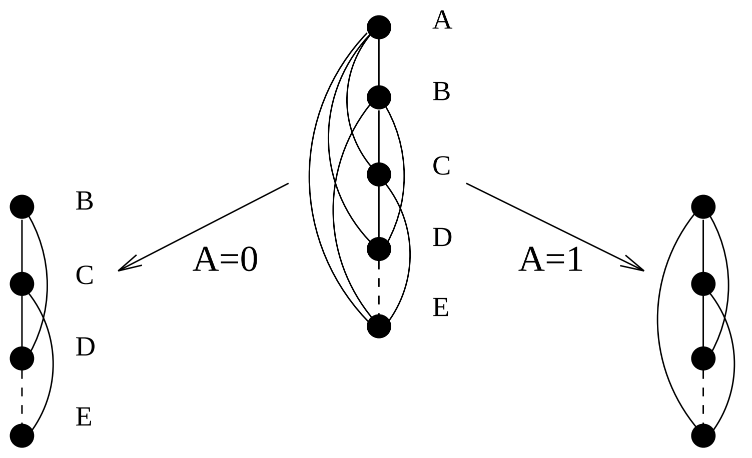


Figure 28: The effect of conditioning on A on the interaction graph of theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$.

The second method of combining DP and DR that we consider uses resolution dynamically during search. We propose a class of hybrid algorithms that select a set of *conditioning* variables (also called a *cutset*), such that instantiating those variables results in a low-width theory tractable for DR [10]. The hybrids run DP on the cutset variables and DR on the remaining ones, thus combining the virtues of both approaches. Like DR, they

[10]This is a generalization of the cycle-cutset algorithm proposed in [20] which transforms the interaction graph of a theory into a tree.

exploit low-w^* structure and produce an output theory that facilitates model generation, while using less space and allowing less average time, like DP.

The description of the hybrid algorithms uses a new notation introduced below. An instantiation of a set of variables $C \subseteq X$ is denoted $I(C)$. The theory φ conditioned on the assignment $I(C)$ is called a *conditional theory* of φ relative to $I(C)$, and is denoted as $\varphi_{I(C)}$. The effect of conditioning on C is deletion of variables in C from the interaction graph. Therefore the *conditional interaction graph* of φ with respect to $I(C)$, denoted $G(\varphi_{I(C)})$, is obtained from the interaction graph of φ by deleting the nodes in C (and all their incident edges). The *conditional width* and *conditional induced width* of a theory φ relative to $I(C)$, denoted $w_{I(C)}$ and $w^*_{I(C)}$, respectively, are the width and induced width of the interaction graph $G(\varphi_{I(C)})$.

For example, Figure 28 shows the interaction graph of theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$ along the ordering $o = (E, D, C, B, A)$ having width and induced width 4. Conditioning on A yields two conditional theories: $\varphi_{A=0} = \{(\neg C \vee E), (B \vee C), (\neg B \vee C \vee D)\}$, and $\varphi_{A=1} = \{(\neg C \vee E), (B \vee E), (\neg B \vee C \vee D)\}$. The ordered interaction graphs of $\varphi_{A=0}$ and $\varphi_{A=1}$ are also shown in Figure 28. Clearly, $w_o(B) = w^*_o(B) = 2$ for theory $\varphi_{A=0}$, and $w_o(B) = w^*_o(B) = 3$ for theory $\varphi_{A=1}$. Note that, besides deleting A and its incident edges from the interaction graph, an assignment may also delete some other edges (e.g., $A = 0$ removes the edge between B and E because the clause $(\neg A \vee B \vee E)$ becomes satisfied).

The conditioning variables can be selected in advance ("statically"), or during the algorithm's execution ("dynamically"). In our experiments, we focused on the dynamic version *Dynamic Conditioning + DR (DCDR)* that was superior to the static one.

Algorithm DCDR(b) guarantees that the induced width of variables that are resolved upon is bounded by b. Given a consistent partial assignment $I(C)$ to a set of variables C, the algorithm performs resolution over the remaining variables having $w^*_{I(C)} < b$. If there are no such variables, the algorithm selects a variable and attempts to assign it a value consistent with $I(C)$. The idea of DCDR(b) is demonstrated in Figure 29 for the theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C \vee D), (\neg A \vee B \vee E \vee D), (\neg B \vee C \vee D)\}$. Assume that we run DCDR(2) on φ. Every variable is initially connected to at least 3 other variables in $G(\varphi)$. As a result, no resolution can be done and a conditioning variable is selected. Assume that A is selected. Assignment $A = 0$ adds the unit clause $\neg A$ which causes unit resolution in $bucket_A$, and produces a new clause $(B \vee C \vee D)$ from $(A \vee B \vee C \vee D)$. The assignment $A = 1$ produces clause $(B \vee E \vee D)$. In Figure 29, the original clauses are shown on the left as a partitioning into buckets. The new clauses are shown on the right, within the corresponding search-tree branches.

Following the branch for $A = 0$ we get a conditional theory $\{(\neg B \vee C \vee D), (B \vee C \vee D),$

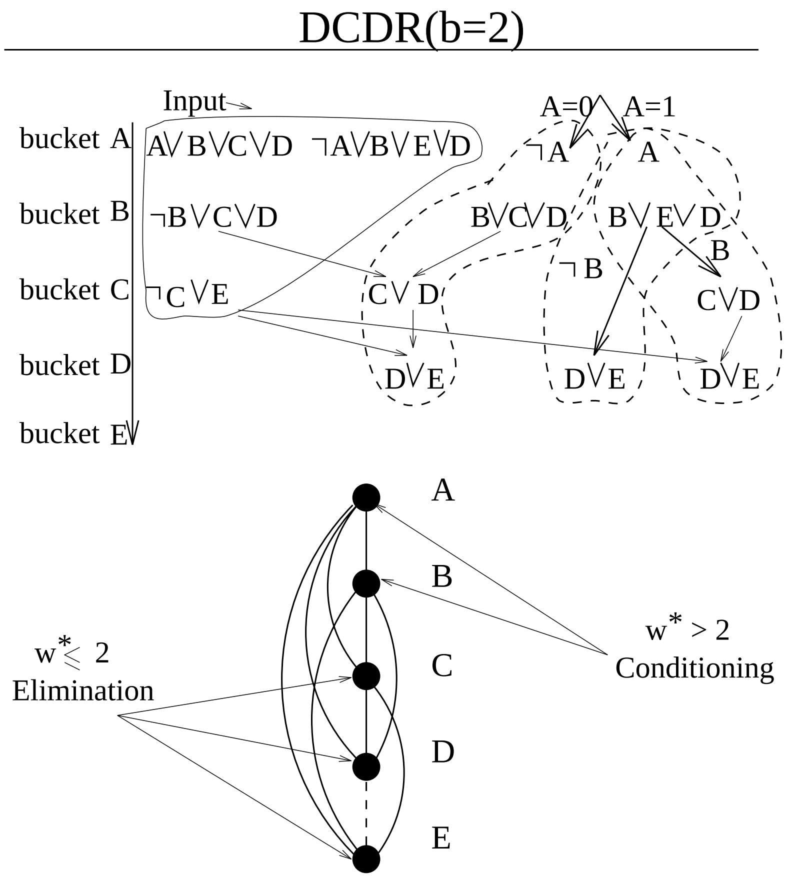


Figure 29: A trace of DCDR(2) on the theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C), (\neg A \vee B \vee E), (\neg B \vee C \vee D)\}$.

$(\neg C \vee E)\}$. Since the degrees of all the variables in the corresponding (conditional) interaction graph are now 2 or less, we can proceed with resolution. We select B, perform resolution in its bucket, and record the resolvent $(C \vee D)$ in $bucket_C$. The resolution in $bucket_C$ creates clause $(D \vee E)$. At this point, the algorithm terminates, returning the assignment $A = 0$, and the conditional directional extension $\varphi \wedge$ $(B \vee C \vee D) \wedge$ $(C \vee D) \wedge$ $(D \vee E)$.

The alternative branch of $A = 1$ results in the conditional theory $\{(B \vee E \vee D), (\neg B \vee C \vee D), (\neg C \vee E)\}$. Since each variable is connected to three other variables, no resolution is possible. Conditioning on B yields the conditional theory $\{(E \vee D), (\neg C \vee E)\}$ when $B = 0$, and the conditional theory $\{(C \vee D), (\neg C \vee E)\}$ when $B = 1$. In both cases, the algorithm terminates, returning $A = 1$, the assignment to B, and the corresponding conditional directional extension.

Algorithm DCDR(b) (Figure 30) takes as an input a propositional theory φ and a parameter b bounding the size of resolvents. Unit propagation is performed first (lines 1-2). If no inconsistency is discovered, DCDR proceeds to its primary activity: choosing between resolution and conditioning. While there is a variable Q connected to at most b other variables in the current interaction graph conditioned on the current assignment, DCDR resolves upon Q (steps 4-9). Otherwise, it selects an unassigned variable (step 10), adds it to the cutset (step 11), and continues recursively with the conditional theory $\varphi \wedge \neg Q$. An unassigned variable is selected using the same dynamic variable ordering heuristic that is used by DP. Should the theory prove inconsistent the algorithm switches to the conditional theory $\varphi \wedge Q$. If both positive and negative assignments to Q are inconsistent the algorithm backtracks to the previously assigned variable. It returns to the previous level of recursion and the corresponding state of φ, discarding all resolvents added to φ after the previous assignment was made. If the algorithm does not find any consistent partial assignment it decides that the theory is inconsistent and returns an empty cutset and an empty directional extension. Otherwise, it returns an assignment $I(C)$ to the cutset C, and the conditional directional extension $E_o(\varphi_{I(C)})$ where o is the variable ordering dynamically constructed by the algorithm. Clearly, the conditional induced width $w^*_{I(C)}$ of φ's interaction graph with respect to o and to the assignment $I(C)$ is bounded by b.

Theorem 12: (DCDR(b) soundness and completeness) *Algorithm DCDR(b) is sound and complete for satisfiability. If a theory φ is satisfiable, any model of φ consistent with the output assignment $I(C)$ can be generated backtrack-free in $O(|E_o(\varphi_{I(C)})|)$ time where o is the ordering computed dynamically by DCDR(b).* $\Box$

Theorem 13: (DCDR(b) complexity) *The time complexity of algorithm DCDR(b) is $O(n2^{\alpha \cdot b+|C|})$, where C is the largest cutset ever conditioned upon by the algorithm, and $\alpha \leq log_2 9$. The space complexity is $O(n \cdot 2^{\alpha \cdot b})$.* $\Box$

The parameter b can be used to control the trade-off between search and resolution. If $b \geq w^*_o(\varphi)$, where o is the ordering used by DCDR(b), the algorithm coincides with DR having time and space complexity exponential in $w^*(\varphi)$. It is easy to show that the ordering generated by DCDR(b) in case of no conditioning yields a min-degree ordering. Thus, given b and a min-degree ordering o, we are guaranteed that DCDR(b) coincides with DR if $w^*_o \leq b$. If $b < 0$, the algorithm coincides with DP. Intermediate values of b allow trading space for time. As b increases, the algorithm requires more space and less time (see also [16]). However, there is no guaranteed worst-case time improvement over DR. It was shown [6] that the size of the smallest *cycle-cutset* C (a set of nodes that breaks all cycles in the interaction graph, leaving a tree, or a forest), and the smallest induced

```
DCDR($\varphi$, $X$, $b$)
Input: A cnf theory $\varphi$ over variables $X$; a bound $b$.
Output: A decision of whether $\varphi$ is satisfiable. If it is, an assignment $I(C)$ to its
conditioning variables, and the conditional directional extension $E_o(\varphi_{I(C)})$.
1. if unit_propagate($\varphi$) = false, return(false);
2. else $X \leftarrow X - \{$ variables in unit clauses $\}$
3. if no more variables to process, return true;
4. else while $\exists Q \in X$ s.t. $degree(Q) \leq b$ in the current graph
5.              resolve over Q
6.              if no empty clause is generated,
7.                 add all resolvents to the theory
8.              else return false
9.              $X \leftarrow X - \{Q\}$
10. Select a variable $Q \in X$; $X \leftarrow X - \{Q\}$
11. $C \leftarrow C \cup \{Q\}$;
12. return( DCDR($\varphi \wedge \neg Q$, $X$, $b$) $\vee$
            DCDR($\varphi \wedge Q$, $X$, $b$) ).
```

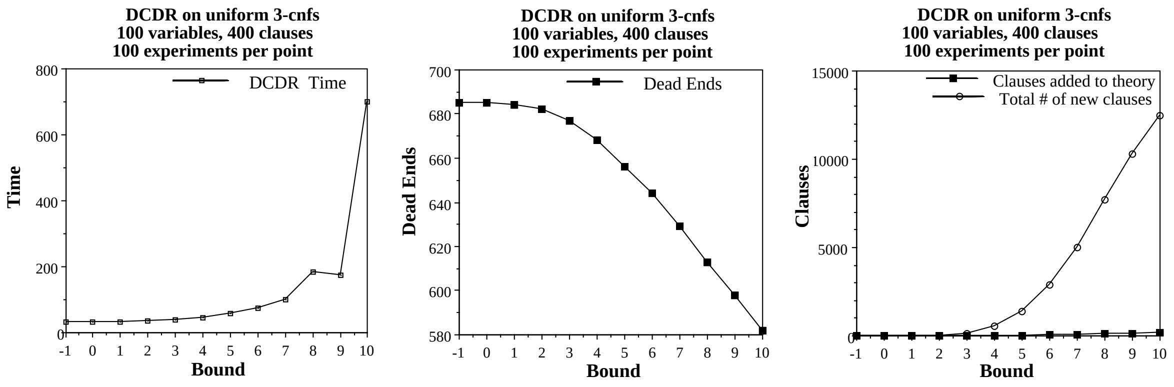
Figure 30: Algorithm DCDR(b).

width, w^*, obey the relation $|C| \geq w^* - 1$. Therefore, for $b = 1$, and for a corresponding cutset C_b, $\alpha \cdot b + |C_b| \geq w^* + \alpha - 1 \geq w^*$, where the left side of this inequality is the exponent that determines complexity of DCDR(b) (Theorem 13). In practice, however, backtracking search rarely demonstrates its worst-case performance and thus the average complexity of DCDR(b) is superior to its worst-case bound as will be confirmed by our experiments.

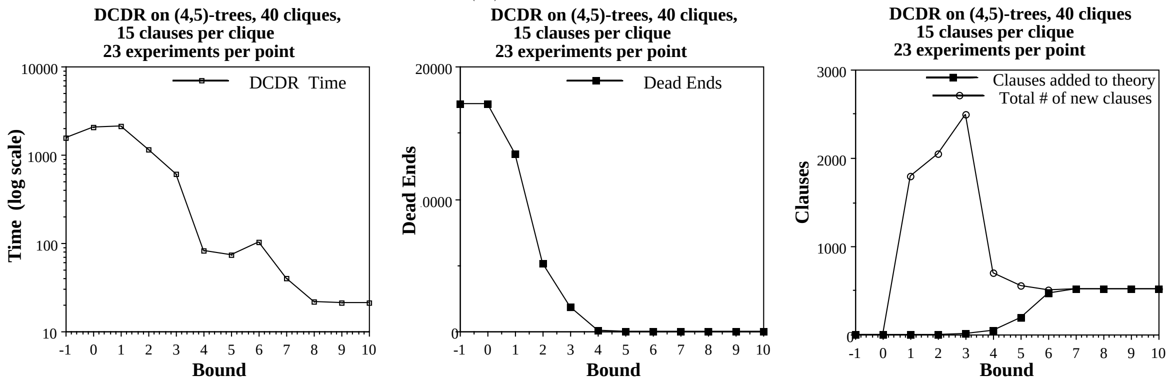
Algorithm DCDR(b) uses the 2-literal-clause ordering heuristic for selecting conditioning variables as used by DP. Random tie-breaking is used for selecting the resolution variables.

8.4 Empirical evaluation of DCDR(b)

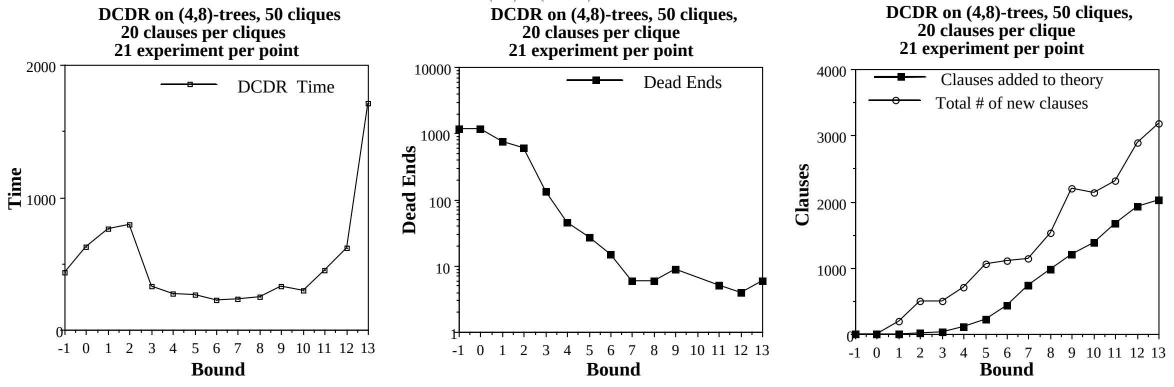
We evaluated the performance of DCDR(b) as a function of b. We tested problem instances in the 50%-satisfiable region (the phase transition region). The results for different b and three different problem structures are summarized in Figures 31-33. Figure 31(a) presents the results for uniform 3-cnfs having 100 variables and 400 clauses. Figures 31(b) and 31(c) focus on $(4,5)$-trees and on $(4,8)$-trees, respectively. We plotted the average time,



(a) uniform 3-cnfs



(b) (4,5)-trees



(c) (4,8)-trees

Figure 31: DCDR(b) on three different classes of 3-cnf problems. Average time, the number of dead-ends, and the number of new clauses are plotted as functions of the parameter b.

the number of dead-ends, and the number of new clauses generated as functions of the bound b (we plot both the total number of generated clauses and the number of clauses actually added to the output theory excluding tautologies and subsumed clauses).

As expected, the performance of DCDR(b) depends on the induced width of the theories. We observed three different patterns:

- On problems having large w^*, such as uniform 3-cnfs in the phase-transition region (see Figure 31), the time complexity of DCDR(b) is similar to DP when b is small. However, when b increases, the CPU time grows exponentially. Apparently, the decline in the number of dead ends is too slow relative to the exponential (in b) growth in the total number of generated clauses. However, the number of new clauses actually *added* to the theory grows slowly. Consequently, the final conditional directional extensions have manageable sizes. We obtained similar results when experimenting with uniform theories having 150 variables and 640 clauses.

- Since DR is equivalent to DCDR(b) whenever b is equal or greater then w^*, for theories having small induced width, DCDR(b) indeed coincides with DR even for small values of b. Figure 31(b) demonstrates this behavior on (4,5)-trees with 40 cliques, 15 clauses per clique, and induced width 6. For $b \geq 8$, the time, the total number of clauses generated, as well as the number of new clauses added to the theory, do not change. With small values of b ($b = 0, 1, 2, 3$), the efficiency of DCDR(b) was sometimes worse than that of DCDR(-1), which is equivalent to DP, due to the overhead incurred by extra clause generation (a more accurate explanation is still required).

- On (k, m)-trees having larger size of cliques (Figure 31(c)), intermediate values of b yielded a better performance than both extremes. DCDR(-1) is still inefficient on structured problems while large induced width made pure *DR* too costly time- and space-wise. For (4,8)-trees, the optimal values of b appear between 5 and 8.

Figure 32 summarizes the results for DCDR(-1), DCDR(5), and DCDR(13) on the three classes of problems. The intermediate bound b=5 seems to be overall more cost-effective than both extremes, b= -1 and b=13.

Figure 33 describes the average number of resolved variables which indicates the algorithm's potential for knowledge compilation. When many variables are resolved upon, the resulting conditional directional extension encodes a larger portion of the models, all sharing the assignment to the cutset variables.

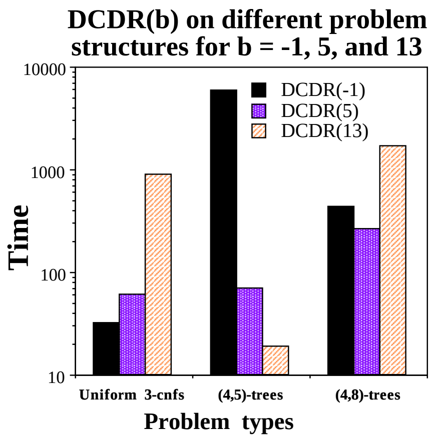


Figure 32: Relative performance of DCDR(b) for $b = -1, 5, 13$ on different types of problems.

9 Related Work

Directional resolution belongs to a family of elimination algorithms first analyzed for optimization tasks in dynamic programming [6] and later used in constraint satisfaction [57, 20] and in belief networks [47]. In fact, DR can be viewed as an adaptation of the constraint-satisfaction algorithm *adaptive consistency* to propositional satisfiability where the project-join operation over relational constraints is replaced by resolution over clauses [20, 24]. Using the same analogy, bounded resolution can be related to bounded consistency-enforcing algorithms, such as arc- path- and i-consistency [48, 30, 14], while bounded directional resolution, BDR(i), parallels directional i-consistency [20, 24]. In-

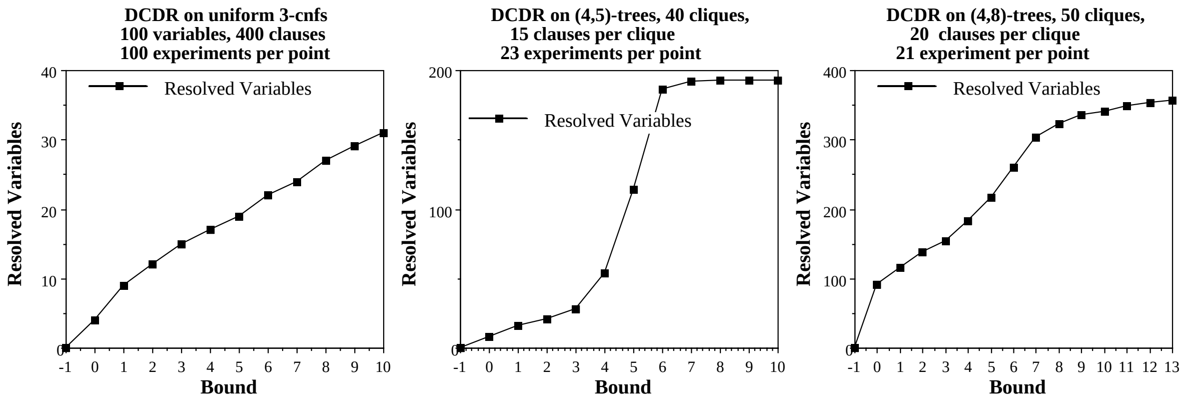


Figure 33: DCDR: the number of resolved variables on different problems.

deed, one of this paper's contributions is transferring constraint satisfaction techniques to the propositional framework.

The recent success of constraint processing which can be attributed to techniques combining search with limited forms of constraint propagation (e.g., forward-checking, MAC, constraint logic programming [41, 36, 56, 43]) that motivated our hybrid algorithms. In the SAT community, a popular form of combining constraint propagation with search is unit-propagation in DP. Our work extends this idea.

The hybrid algorithm BDR-DP(i), initially proposed in [23], corresponds to applying directional i-consistency prior to backtracking search for constraint processing. This approach was empirically evaluated for some constraint problems in [19]. However, those experiments were restricted to small and relatively easy problems, for which only a very limited amount of preprocessing was cost-effective. The presented experiments with BDR-DP(i) suggest that the results in [19] were too preliminary and that the idea of preprocessing before search is viable and should be further investigated.

Our second hybrid algorithm, DCDR(b), proposed first in [53], generalizes the cycle-cutset approach that was presented for constraint satisfaction [13] using static variable ordering. This idea of alternating search with bounded resolution was also suggested and evaluated independently by van Gelder in [38], where a generalization of unit resolution known as *k-limited resolution* was proposed. This operation requires that the operands and the resolvent have at most k literals each. The hybrid algorithm proposed in [38] computes *k-closure* (namely, it applies k-limited resolution iteratively and eliminates subsumed clauses) between branching steps in DP-backtracking. This algorithm, augmented with several branching heuristics, was tested for k=2 (the combination called $2cl$ algorithm), and demonstrated its superiority to DP, especially on larger problems. Algorithm DCDR(b) computes a *subset* of b-closure between its branching steps [11]. In this paper, we study the impact of b on the effectiveness of hybrid algorithms over different problem structures, rather than focus on a fixed b.

The relationship between clausal tree-clustering and directional resolutions extends the known relationship between variable elimination and the tree-clustering compilation scheme that was presented for constraint satisfaction in [21] and was extended to probabilistic frameworks in [15].

[11]DCDR(b) performs resolution on variables that are connected to at most b other variables; therefore, the size of resolvents is bounded b. It does not, however, resolve over the variables having degree higher than b in the conditional interaction graph, although such resolutions can sometimes produce clauses of size not larger than b.

10 Summary and Conclusions

The paper compares two popular approaches to solving propositional satisfiability, backtracking search and resolution, and proposes two parameterized hybrid algorithms. We analyze the complexity of the original resolution-based Davis-Putnam algorithm, called here directional resolution (DR)), as a function of the induced width of the theory's interaction graph. Another parameter called diversity provides an additional refinement for tractable classes. Our empirical studies confirm previous results showing that on uniform random problems DR is indeed very inefficient. However, on structured problems such as k-tree embeddings, having bounded induced width, directional resolution outperforms the popular backtracking-based Davis-Putnam-Logemann-Loveland Procedure (DP). We also emphasize the knowledge-compilation aspects of directional resolution as a procedure for tree-clustering. We show that it generates all prime implicates restricted to cliques in the clique-tree.

The two parameterized hybrid schemes, BDR-DP(i) and DCDR(b), allow a flexible combination of backtracking search with directional resolution. Both schemes use a parameter that bounds the size of the resolvents recorded. The first scheme, BDR-DP(i), uses bounded directional resolution BDR(i) as a preprocessing step, recording only new clauses of size i or less. The effect of the bound was studied empirically over both uniform and structured problems, observing that BDR-DP(i) frequently achieves its optimal performance for intermediate levels of i, outperforming both DR and DP. We also believe that the transition from i=3 to i=4 is too sharp and that intermediate levels of preprocessing are likely to provide even better trade-off. Encouraging results are obtained for BDR-DP(i) on DIMACS benchmark, where the hybrid algorithm easily solves some of the problems that were hard both for DR and DP.

The second hybrid scheme uses bounded resolution *during* search. Given a bound b, algorithm DCDR(b) instantiates a dynamically selected subset of conditioning variables such that the induced width of the resulting (conditional) theory and therefore the size of the resolvents recorded does not exceed b. When $b \leq 0$, $DCDR(b)$ coincides with DP, while for $b \geq w^*_o$ (on the resulting ordering o) it coincides with directional resolution. For intermediate b, DCDR(b) was shown to outperform both extremes on intermediate-w^* problem classes.

For both schemes selecting the bound on the resolvent size allows a flexible scheme that can be adapted to the problem structure and to computational resources. Our current "rule of thumb" for DCDR(b) is to use small b when w^* is large, relying on search, large b when w^* is small, exploiting resolution, and some intermediate bound for intermediate w^*. Additional experiments are necessary to further demonstrate the spectrum of optimal hybrids relative to problem structures.

References

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

[2] R. Bayardo and D. Miranker. A complexity analysis of space-bound learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 298–304, 1996.

[3] R.J. Bayardo and R.C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of AAAI-97*, pages 203 –208, 1997.

[4] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal jnmction trees. In *Uncertainty in AI (UAI-96)*, pages 81–89, 1996.

[5] R. Ben-Eliyahu and R. Dechter. Default reasoning using classical logic. *Artificial Intelligence*, 84:113–150, 1996.

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

[7] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the *Really* Hard Problems Are. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 331–337, 1991.

[8] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.

[9] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 21–27, 1993.

[10] J.M. Crawford and A.B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of AAAI-94, Seattle, WA*, pages 1092 – 1097, 1994.

[11] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Communications of the ACM*, 5:394–397, 1962.

[12] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960.

[13] R. Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[14] R. Dechter. Constraint networks. In *Encyclopedia of Artificial Intelligence*, pages 276–285. John Wiley & Sons, 2nd edition, 1992.

[15] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 211–219, 1996.

[16] R. Dechter. Topological parameters for time-space tradeoffs. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 220–227, 1996.

[17] R. Dechter and A. Itai. Finding all solutions if you can find one. In *UCI Technical report R23, 1992. Also in the Proceedings of the Workshop on tractable reasoning, AAAI-92*, 1992.

[18] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *International Joint Conference on Artificial Intelligence*, pages 271–277, 1989.

[19] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

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

[21] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[22] R. Dechter and J. Pearl. Directed constraint networks: A relational framework for causal models. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), Sidney, Australia*, pages 1164–1170, 1991.

[23] R. Dechter and I. Rish. Directional resolution: the Davis-Putnam procedure, revisited. In *Proceedings of KR-94*, 1994.

[24] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, pages 283–308, 1997.

[25] A. del Val. A new method for consequence finding and compilation in restricted languages. In *Proceedings of AAAI-99*, 1999.

[26] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multi-commodity flow. *SIAM Journal on Computing*, 5:691–703, 1976.

[27] Y. El Fattah and R. Dechter. Diagnosing tree-decomposable circuits. In *International Joint Conference of Artificial Intelligence (IJCAI-95)*, pages 1742–1748, Montreal, Canada, August 1995.

[28] Y. El Fattah and R. Dechter. An evaluation of structural parameters for probabilistic reasoning: results on benchmark circuits. In *UAI96*, pages 244–251, Portland, Oregon, August 1996.

[29] J. Franco and M. Paul. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Appl. Math.*, 5:77 – 87, 1983.

[30] E. C. Freuder. Synthesizing constraint expressions. *Communication of the ACM*, 21(11):958–965, 1978.

[31] D. Frost and R. Dechter. Dead-end driven learning. In *AAAI-94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 294–300, 1994.

[32] D. Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. In *Proc. of National Conference on Artificial Intelligence (AAAI97)*, pages 327–333, 1997.

[33] D. H. Frost. Algorithms and heuristics for constraint satisfaction problems. Technical report, Phd thesis, Information and Computer Science, University of California, Irvine, California, 1997.

[34] Daniel Frost and Rina Dechter. In search of the best constraint satisfaction search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 1994.

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

[36] J. Gaschnig. A General Backtrack Algorithm That Eliminates Most Redundant Tests. In *Proceedings of the International Joint Conference on Artificial Intelligence*, page 247, 1977.

[37] J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie Mellon University, 1979.

[38] A. Van Gelder and Y. K. Tsuji. Satisfiability testing with more reasoning and less guessing. In *David, Johnson and Michael A. Trick, editors, Cliques, Coloring and Satisfiability*, 1996.

[39] A. Goerdt. Davis-Putnam resolution versus unrestricted resolution. *Annals of Mathematics and Artificial Intelligence*, 6:169–184, 1992.

[40] A. Goldberg, P. Purdom, and C. Brown. Average time analysis of simplified Davis-Putnam procedures. *Information Processing Letters*, 15:72–75, 1982.

[41] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[42] J.N. Hooker and V. Vinay. Branching rules for satisfiability. In *Third International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida*, 1994.

[43] J. Jaffar and J. Lassez. Constraint logic programming: A survey. *Journal of Logic Programming*, 19(20):503–581, 1994.

[44] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167 –187, 1990.

[45] K. Kask and R. Dechter. Gsat and local consistency. In *Proceedings of IJCAI-95*, pages 616–622, 1995.

[46] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, 1996.

[47] S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[48] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[49] David Mitchell, Bart Selman, and Hector Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 459–465, 1992.

[50] P. Prosser. Hybrid algorithms for constraint satisfaction problems. *Computational Intelligence*, 9(3):268–299, 1993.

[51] Patrick Prosser. BM + BJ = BMJ. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications*, pages 257–262, 1983.

[52] I. Rish. *Efficient reasoning in graphical models.* PhD thesis, 1999.

[53] I. Rish and R. Dechter. To guess or to think? hybrid algorithms for SAT (extended abstract). In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP96)*, 1996.

[54] I. Rish and D. Frost. Statistical analysis of backtracking on inconsistent CSPs. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP97)*, 1997.

[55] N. Robertson and P. Seymour. Graph minor. xiii. the disjoint paths problem. *Combinatorial Theory, Series B*, 63:65–110, 1995.

[56] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *ECAI-94*, pages 125–129, Amsterdam, 1994.

[57] R. Seidel. A new method for solving constraint satisfaction problems. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence (IJCAI-81), Vancouver, Canada*, pages 338–342, 1981.

[58] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of AAAI94*, pages 337–343, 1994.

[59] Bart Selman, Hector Levesque, and David Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

[60] Barbara M. Smith and M. E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:155–181, 1996.

[61] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computation.*, 13(3):566–579, 1984.

Appendix A: Proofs

Theorem 2: (model generation)
Given $E_o(\varphi)$ of a satisfiable theory φ, the procedure find-model *generates a model of φ backtrack-free, in time $O(|E_o(\varphi)|)$.*
Proof: Suppose the model-generation process is not backtrack-free. Namely, suppose there exists a truth assignment $q_1, ..., q_{i-1}$ for the first $i-1$ variables in the ordering $o = (Q_1, ..., Q_n)$ that satisfies all the clauses in the buckets of Q_1,..., Q_{i-1}, but cannot be extended by any value of Q_i without falsifying some clauses in $bucket_i$. Let α and β be two clauses in the bucket of Q_i that cannot be satisfied simultaneously, given the assignment $q_1, ..., q_{i-1}$. Clearly, Q_i appears negatively in one clause and positively in the other. Consequently, while being processed by DR, α and β should be resolved, resulting in a clause γ that must reside now in a $bucket_j$, $j < i$. That clause can not allow the partial model $q_1, ..., q_i$, which contradicts our assumption. Since the model-generation is backtrack-free, it takes $O(|E_o(\varphi)|)$ time consulting all the buckets. □

Theorem 3: (complexity)
Given a cnf theory φ and an ordering o, the time complexity of algorithm DR is $O(n \cdot |E_o(\varphi)|^2)$ where n is the number of variables.
Proof: There are at most n buckets, each containing no more clauses than the output directional extension. The number of resolution operations in a bucket does not exceed the number of all possible pairs of clauses, which is quadratic in the size of the bucket. This yields the complexity $O(n \cdot |E_o(\varphi)|^2)$. □

Lemma 1: *Given a cnf theory φ and an ordering o, $G(E_o(\varphi))$ is a subgraph of $I_o(G(\varphi))$.*
Proof: The proof is by induction on the variables along ordering $o = (Q_1, ..., Q_n)$. The induction hypothesis is that all the edges incident to $Q_n, ..., Q_i$ in $G(E_o(\varphi))$ appear also in $I_o(G(\varphi))$. The claim is clearly true for Q_n. Assume that the claim is true for $Q_n, ..., Q_i$; as we show, this assumption implies that if (Q_{i-1}, Q_j), $j < i-1$, is an edge in $G(E_o(\varphi))$, then it also belongs to $I_o(G(\varphi))$. There are two cases: either Q_{i-1} and Q_j initially appeared in the same clause of φ and so are connected in $G(\varphi)$ and, therefore, also in $I_o(G(\varphi))$, or a clause containing both variables was added during directional resolution. In the second case, that clause was obtained while processing some bucket Q_t, where $t > i-1$. Since Q_{i-1} and Q_j appeared in the bucket of Q_t, each must be connected to Q_t in $G(E_o(\varphi))$ and, by the induction hypothesis, each will also be connected to Q_t in $I_o(G(\varphi))$. Since Q_{i-1} and Q_j are parents of Q_t, they must be connected in $I_o(G(\varphi))$. □

Lemma 2: *Given a theory φ and an ordering $o = (Q_1, ..., Q_n)$, if Q_i has at most k parents in the induced graph along o, then the bucket of a variable Q_i in $E_o(\varphi)$ contains no more than 3^{k+1} clauses.*

Proof: Given a clause α in the bucket of Q_i, there are three possibilities for each parent P: either P appears in α, or $\neg P$ appears in α, or neither of them appears in α. Since Q_i also appears in α, either positively or negatively, there are no more than $2 \cdot 3^k < 3^{k+1}$ different clauses in the bucket. $\square$

Theorem 4: *(complexity of DR)*
Given a theory φ and an ordering of its variables o, the time complexity of algorithm DR along o is $O(n \cdot 9^{w^*_o})$, and the size of $E_o(\varphi)$ is at most $n \cdot 3^{w^*_o+1}$ clauses, where w^*_o is the induced width of φ's interaction graph along o.

Proof: The result follows from lemmas 1 and 2. The interaction graph of $E_o(\varphi)$ is a subgraph of $I_o(G)$ (lemma 1), and the size of theories having $I_o(G)$ as their interaction graph is bounded by $n \cdot 3^{w^*(o)+1}$ (lemma 2). The time complexity of algorithm DR is bounded by $O(n \cdot |bucket_i|^2)$, where $|bucket_i|$ is the size of the largest bucket. By lemma 2, $|bucket_i| = O(3^{w^*(o)})$. Therefore, the time complexity is $O(n \cdot 9^{w^*(o)})$. $\square$

Theorem 7: *Given a theory φ defined on variables $Q_1, ..., Q_n$, such that each symbol Q_i either (a) appears only negatively (only positively), or (b) it appears in exactly two clauses, then $div^*(\varphi) \leq 1$ and φ is tractable.*

Proof: The proof is by induction on the number of variables. If φ satisfies either (a) or (b), we can select a variable Q with the diversity of at most 1 and put it last in the ordering. Should Q have zero diversity (case a), no clause is added. If it has diversity 1 (case b), then at most one clause is added when processing its bucket. Assume the clause is added to the bucket of Q_j. If Q_j is a single-sign symbol, it will remain so. The diversity of its bucket will be zero. Otherwise, since there are at most two clauses containing Q_j, and one of these was in the bucket of Q_n, the current bucket of Q_j (after processing Q_n) cannot contain more than two clauses. The diversity of Q_j is therefore 1. We can now assume that after processing $Q_n, ..., Q_i$ the induced diversity is at most 1, and can also show that processing Q_{i-1} will leave the diversity at most 1. $\square$

Theorem 8: *Algorithm* min-diversity *generates a minimal diversity ordering of a theory. Its time complexity is $O(n^2 \cdot c)$, where n is the number of variables and c is the number of clauses in the input theory.*
Proof: Let o be an ordering generated by the algorithm and let Q_i be a variable whose diversity equals the diversity of the ordering. If Q_i is pushed up, its diversity can only increase. When it is pushed down, it must be replaced by a variable whose diversity is equal to or higher than the diversity of Q_i. Computing the diversity of a variable takes $O(c)$ time, and the algorithm checks at most n variables in order to select the one with the smallest diversity at each of n steps. This yields the total $O(n^2 \cdot c)$ complexity. $\square$

Lemma 3: *Given a theory φ, let $T = TCC(\varphi)$ be a clause-based join-tree of φ, and let C be a clique in T. Then, there exist an ordering o that can start with any ordering of the variables in C, such that $E_o(\varphi) \subseteq TCC(\varphi)$.*
Proof: Once the join-tree structure is created, the order of processing the cliques (from leaves to root) is dependent on the identity of the root clique. Since processing is applied once in each direction, the resulting join-tree is invariant to the particular rooted tree selected. Consequently, we can assume that the clique C is the root, and it is the last to be processed in the backwards phase of DR-TC. Let o_C be a tree-ordering of the cliques that starts with C, and let o be a possible ordering of the variables that is consistent with o_C. Namely for every two variables X and Y if there are two cliques C_1 and C_2 s.t. $X \in C_1$ and $Y \in C_2$ and C_1 is ordered before C_2 in o_C, then X should appear before Y in o. It is easy to see that directional-resolution applied to φ using o (in reversed order), generates a subset of the resolvents that are created by the backwards phase of DR-TC using o_C. Therefore $E_o(\varphi) \subseteq TCC_o(\varphi)$. $\square$

Theorem 11: *Let φ be a theory and $T = TCC_o(\varphi)$ be a clause-based join-tree of φ. Then for every clique $C \in T$, $prime_\varphi(C) \subseteq TCC(\varphi)$.*
Proof: Consider an arbitrary clique C. Let $P_1 = prime_\varphi(C)$ and let $P_2 = TCC(\varphi)$. We want to show that $P_1 \subseteq P_2$. If not, there exists a prime implicate $\alpha \in P_1$, defined on subset $S \subseteq C$, that was not derived by DR-TC. Assume that C is the root of the join-tree computed by DR-TC. Let o be an ordering consistent with this rooted tree that starts with the variables in S. From lemma 3 it follows that the directional extension $E_o(\varphi)$ is contained in $TCC(\varphi)$, so that any model along this ordering can be generated in a

backtrack-free manner by consulting $E_o(\varphi)$ (Theorem 2). However, nothing will prevent model-generation from assigning S the no-good $\neg\alpha$ (since it is not available, no subsuming clauses exist). This assignment leads to a deadend, contradicting the backtrack-free property of the directional extension. $\Box$

Corollary 3: *Given a theory φ and given $TCC_o(\varphi)$ of some ordering o the following properties hold:*

1. *The theory φ is satisfiable if and only if $TCC(\varphi)$ does not contain an empty clause.*
2. *If $T = TCC(\varphi)$ for some φ then entailment of any clause whose variables are contained in a single clique can be decided in linear time.*
3. *Entailment of a clause α from φ can be decided in $O(exp(w^* + 1))$ time in φ, when $w^* + 1$ is the maximal clique size.*
4. *Checking if a new clause is consistent with φ can be done in time linear in T.*

Proof:

1. If no empty clause is encountered, the theory is satisfiable and vice-versa.
2. Entailment of a clause α whose variables are contained in clique C_i can be decided by scanning the compiled φ_i^*. If no clause subsuming α exists, then α is not entailed by φ.
3. Entailment of an arbitrary clause can be checked by placing the negation of each literal in the largest-index clique that contains the corresponding variable, and repeating the first pass of DR-TC over the join-tree. The clause is entailed if and only if the empty clause was generated, which may take $O(exp(w^*))$ time.
4. Consistency of a clause α is decided by checking the entailment of its negated literals. α is not consistent with φ if and only if the theory entails each of the negated literals of α. Entailment of each negated literal can be decided in linear time.

$\Box$

Theorem 12: (DCDR(b) soundness and completeness)

Algorithm DCDR(b) is sound and complete for satisfiability. If a theory φ is satisfiable, any model of φ consistent with the output assignment $I(C)$ can be generated backtrack-free in $O(|E_o(\varphi_{I(C)})|)$ time, where o is the ordering computed dynamically by DCDR(b).

Proof: Given an assignment $I(C)$, DCDR(b) is equivalent to applying DR to the theory $\varphi_{I(C)}$ along ordering o. From Theorem 2 it follows that any model of $\varphi_{I(C)}$ can be found in a backtrack-free manner in time $O(|E_o(\varphi_{I(C)})|)$. $\square$

Theorem 13: (DCDR(b) complexity) *The time complexity of algorithm DCDR(b) is $O(n2^{\alpha \cdot b+|C|})$, where C is the largest cutset ever instantiated by the algorithm, and $\alpha \leq log_2 9$. The space complexity is $O(n \cdot 2^{\alpha \cdot b})$.*

Proof: Given a cutset assignment, the time and space complexity of resolution steps within DCDR(b) is bounded by $O(n \cdot 9^b)$ (see theorem 4). Since in the worst-case backtracking involves enumerating all possible instantiations of the cutset variables C in $O(2^{|C|})$ time and $O(|C|)$ space, the total time complexity is $O(n \cdot 9^b \cdot 2^{|C|}) = O(n \cdot 2^{\alpha \cdot b+|C|})$, where C is the largest cutset ever instantiated by the algorithm, and $\alpha \leq log_2 9$. The total space complexity is $O(|C| + n \cdot 9^b) = O(n \cdot 9^b)$. $\square$