Directional Resolution: The Davis-Putnam Procedure, Revisited *

Rina Dechter and Irina Rish

Information and Computer Science

University of California, Irvine

dechter@ics.uci.edu, *irinar@ics.uci.edu*

Abstract

The paper presents an algorithm called *directional resolution*, a variation on the original Davis-Putnam algorithm, and analyzes its worst-case behavior as a function of the topological structure of propositional theories. The concepts of *induced width* and *diversity* are shown to play a key role in bounding the complexity of the procedure. The importance of our analysis lies in highlighting structure-based tractable classes of satisfiability and in providing theoretical guarantees on the time and space complexity of the algorithm. Contrary to previous assessments, we show that for many theories directional resolution could be an effective procedure. Our empirical tests confirm theoretical prediction, showing that on problems with a special structure, namely *k-tree embeddings* (e.g. *chains*, *(k,m)-trees*), directional resolution greatly outperforms one of the most effective satisfiability algorithms known to date, the popular Davis-Putnam procedure. Furthermore, combining a bounded version of directional resolution with the Davis-Putnam procedure results in an algorithm superior to either components.

*This work was partially supported by NSF grant IRI-9157636, by Air Force Office of Scientific Research grant AFOSR 900136, by Toshiba of America, and by a Xerox grant.

1 Introduction

In 1960, Davis and Putnam presented a resolution algorithm for determining propositional satisfiability [6]. They proved that a restricted amount of resolution performed systematically along some ordering of the variables in a propositional theory is sufficient for deciding satisfiability. This algorithm, in its original form, has received limited attention, and analyses of its performance have emphasized its worst-case exponential behavior [12, 14], while neglecting the algorithm's virtues. This happened, in our view, because the algorithm was immediately overshadowed by a competitor with nearly the same name: *The Davis-Putnam Procedure*. This competing algorithm, proposed in 1962 by Davis, Logemann, and Loveland [5], searches through the space of possible truth assignments while performing unit resolution until quiesience at each step. We will refer to the first algorithm as *DP-elimination* and to the second as *DP-backtracking*. The latter was presented in [5] as a minor syntactic change to the first: the *elimination rule* (rule *III* in [6]) in DP-elimination was replaced by the *splitting rule* (rule *III'* in [5]) in order to avoid the memory explosion encountered when empirically testing DP-elimination. By refraining from an explicit analysis of this exchange (beyond the short comment on memory explosion), the authors of [5] may have left the impression that the two algorithms are basically identical. Indeed, from then on, most work on the Davis-Putnam procedure quotes the backtracking version [15, 19], wrongly suggesting that this is the algorithm presented in [6].

In this paper, we wish to "revive" the DP-elimination algorithm by studying its virtues theoretically and by subjecting it to a more extensive empirical testing. First, we show that, in addition to determining satisfiability, the algorithm generates an equivalent theory that facilitates model generation and query processing. Consequently, it may be better viewed as a knowledge compilation algorithm. Second, we offset the known worst-case exponential complexities [12, 14] by showing the tractability of DP-elimination for many known tractable classes of satisfiability and constraint satisfaction problems (e.g., 2-cnfs, Horn clauses, causal theories and theories having a bounded induced width [7, 8]). Third, we introduce a new parameter, called *diversity*, that gives rise to new tractable classes.

On the empirical side, we qualify prior empirical tests in [5] by showing that for uniform random propositional theories DP-backtracking outperforms DP-elimination by far. However, for a class of instances having a special structure (embeddings in *k-trees*, for example *chains* and *(k,m)-trees*) DP-elimination outperforms DP-backtracking by several orders of magnitude. Also, a restricted version of DP-elimination, called *bounded directional resolution*, used as a preprocessing for DP-backtracking, improves the performance of the latter both on uniform and structured problems. Empirical results show that the combined algorithm called *BDR-DP* outperforms both DP-elimination and DP-backtracking.

2 Definition and preliminaries

We denote propositional symbols, also called *variables*, by uppercase letters $P, Q, R, ...$, propositional literals (e.g., $P, \neg P$) by lowercase letters $p, q, r, ...$, and disjunctions of literals, or *clauses*, by $\alpha, \beta, ...$ For instance, $\alpha = (P \vee Q \vee R)$ is a clause. We will sometime denote the clause $(P \vee Q \vee R)$ by $\{P, Q, R\}$. A *unit clause* is a clause with only one literal. The notation $(\alpha \vee T)$ will be used as a shorthand for the disjunction $(P \vee Q \vee R \vee T)$, and $\alpha \vee \beta$ denotes the clause whose literals appear in either α or β. The *resolution* operation over two clauses $(\alpha \vee Q)$ and $(\beta \vee \neg Q)$ results in a clause $(\alpha \vee \beta)$, thus eliminating Q. *Unit resolution* is a resolution operation when one of the clauses is a unit clause. A formula φ in conjunctive normal form (*cnf*) is represented as a set $\{\alpha_1, ..., \alpha_t\}$ denoting the conjunction of clauses $\alpha_1, ..., \alpha_t$. The set of *models* of a formula φ is the set of all truth assignments to its symbols that satisfy φ. A clause α is *entailed* by φ, $\varphi \models \alpha$, iff α is true in all models of φ. A *Horn formula* is a cnf formula whose clauses have at most one positive literal. In a *definite Horn formula*, each clause has exactly one positive literal. A clause is positive if it contains only positive literals and is negative if it contains negative literals only. A k-cnf formula is one whose clauses are all of length k or less.

3 DP-elimination – Directional Resolution

DP-elimination [6] is an ordering-based restricted resolution that can be described as follows. Given an arbitrary ordering of the propositional variables, we assign to each clause the index of the highest ordered literal in that clause. Then we resolve only 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 for future resolution. DP-elimination also includes additional steps, one forcing unit resolution whenever possible and another dealing with the variables that appear only negatively (called *all-negative*) or only positively (called *all-positive*). All-positive (all-negative) variables are assigned the value "true" ("false") and all clauses containing those variables are deleted from the theory. There are many other intermediate steps that can be introduced between the basic steps of eliminating the highest indexed variable (e.g., subsumption elimination). However, in this paper we focus on the ordered elimination step and refer to auxiliary steps only when necessary. We are interested not merely in achieving refutation, but also in the sum total of the clauses accumulated by this process, which constitutes an equivalent theory with useful computational features. Algorithm *directional resolution* (DR) (the core of DP-elimination) is described in Figure 1. We call its output theory, $E_d(\varphi)$, the *directional extension* of φ.

The algorithm can be conveniently described using the notion of *buckets* partitioning the set of clauses in φ. Given an ordering $d = Q_1, ...Q_n$, the bucket for Q_i, $bucket_i$, contains all the clauses containing Q_i, that do not contain any symbol higher in the ordering. Given the theory φ, algorithm *directional resolution* processes the buckets in a reverse order of d. When processing $bucket_i$, it resolves over Q_i all possible pairs of clauses in the bucket and inserts the resolvents into the appropriate lower buckets.

Theorem 1: (model generation)

Let φ be a cnf formula, $d = Q_1, ..., Q_n$ an ordering, and $E_d(\varphi)$ its directional extension. Then, if the extension is not empty, any model of φ can be generated in time $O(|E_d(\varphi)|)$ in a backtrack-free manner, consulting $E_d(\varphi)$, as follows: Step 1. Assign to Q_1 a truth value that

directional-resolution

Input: A cnf theory φ, an ordering $d = Q_1, ..., Q_n$ of its variables.

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

1. *Initialize:* generate an ordered partition of the clauses, $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the clauses whose highest literal is Q_i.

2. For $i = n$ to 1 do:

3. Resolve each pair $\{(\alpha \vee Q_i), (\beta \vee \neg Q_i)\} \subseteq bucket_i$. If $\gamma = \alpha \vee \beta$ is empty, return $E_d(\varphi) = \emptyset$, the theory is not satisfiable; else, determine the index of γ and add it to the appropriate bucket.

4. End-for.

5. Return $E_d(\varphi) \Longleftarrow \bigcup_i bucket_i$.

Figure 1: Algorithm *directional resolution*

is consistent with clauses in $bucket_1$ *(if the bucket is empty, assign* Q_1 *an arbitrary value); Step i. After assigning values to* $Q_1, ..., Q_{i-1}$*, assign a value to* Q_i*, so that together with the previous assignments it will satisfy all clauses in* $bucket_i$*.*

Proof: Suppose the model generation process is not backtrack-free, i.e there exists a truth assignments $q_1, ..., q_{i-1}$ for the first $i-1$ symbols that satisfies all the clauses in the buckets of Q_1,..., Q_{i-1}, but cannot be extended by any truth 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 clash, i.e. cannot be satisfied simultaneously, given the assignment $q_1, ..., q_{i-1}$. Clearly, α and β contain Q_i with opposite signs; in one Q_i appears negatively and in the other positively. Consequently, while being processed by directional-resolution, α and β should have been resolved upon, thus resulting in a clause that must appear now in a $bucket_j$, $j < i$. Such a clause, if existed, would not have allowed the partial model $q_1, ..., q_i$, thus leading to a contradiction. □

Corollary 1: *[6] A theory has a non-empty directional extension iff it is satisfiable.* □

Clearly, the effectiveness of directional resolution both for satisfiability and for subsequent query processing depends on the the size of its output theory $E_d(\varphi)$.

Theorem 2: (complexity)

Given a theory φ and an ordering d of its propositional symbols, the time complexity of algorithm directional resolution is $O(n \cdot |E_d(\varphi)|^2)$, where n is the number of propositional symbols in the language.

Proof: There are at most n buckets, each containing no more clauses than the directional extension of the theory, and resolving pairs of clauses in a bucket is quadratic in its size. □

The bound above, although it could be loose, demonstrates the dependence of the algorithm's complexity on the size of its resulting output. Once $E_d(\varphi)$ is compiled, determining the entailment of a single literal involves checking the bucket of that literal first. If the literal appears there as a unit clause, it is entailed; if not, the negation of that literal must be added to the theory and the algorithm must be restarted from that bucket. If the empty clause is generated, the literal is entailed. To determine the entailment of an arbitrary clause, the negation of each literal in that clause must be added to the corresponding bucket; then the processing must be restarted from the highest of those buckets. Therefore, in knowledge bases with queries involving a restricted subset of the alphabet the ordering for directional resolution should start with the symbols of that subset. In summary,

Theorem 3: *(*entailment*)*

Given a directional extension $E_d(\varphi)$ and a constant c, the entailment of clauses involving only the first c symbols in d is polynomial in the size of $E_d(\varphi)$. □

4 Tractable classes

Consider the following two examples demonstrating the effect of ordering on $E_d(\varphi)$.

Example 1: Let $\varphi_1 = \{(B, A), (C, \neg A), (D, A), (E, \neg A)\}$. For the ordering $d_1 = (E, B, C, D, A)$, all clauses are initially contained in $bucket(A)$ (highest in the ordering). All other buckets are empty. Following the application of directional resolution along d_1, we get (note that processing is in the reverse order of d): $bucket(D) = \{(C, D), (D, E)\}$, $bucket(C) = \{(B, C)\}$,

$bucket(B) = \{(B, E)\}$. On the other hand, the directional extension along the ordering $d_2 = (A, B, C, D, E)$ is identical to the input theory, and each bucket contains at most one clause.

Example 2: Consider the theory $\varphi_2 = \{(\neg A, B), (A, \neg C), (\neg B, D), (C, D, E)\}$. The directional extensions of φ along the ordering $d_1 = (A, B, C, D, E)$ and $d_2 = (D, E, C, B, A)$ are $E_{d_1}(\varphi) = \varphi$ and $E_{d_2}(\varphi) = \varphi \cup \{(B, \neg C), (\neg C, D), (E, D)\}$, respectively.

In Example 1, A appears in all clauses; hence, it potentially can generate new clauses when resolved upon, unless it is processed last (i.e., appears first in the ordering), as in d_2. This shows that the interactions among clauses play an important role in the effectiveness of the algorithm and may suggest orderings that yield smaller extensions. In Example 2, on the other hand, all symbols have the same type of interaction, each (except E) appearing in two clauses. Nevertheless, D appears positive in both clauses and consequently will not be resolved upon; hence, it can be processed first. Subsequently, B and C appear only negatively in the remaining theory and can be processed without generating new clauses. In the following, we will provide a connection between the algorithm's complexity and two parameters: a topological parameter, called *induced width*, and a syntactic parameter, called *diversity*.

Note that directional resolution is tractable for 2-cnf theories in all orderings, since 2-cnf 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, unrestricted resolution is also tractable). Clearly, this algorithm is not the most effective one for satisfiability of 2-cnfs. Satisfiability for these theories can be decided in linear time [10]. However, as noted earlier, DR achieves more than satisfiability, it compiles a theory that allows model generation in linear time. We summarize:

Theorem 4: *If φ is a 2-cnf theory, then algorithm directional resolution will produce a directional extension of size $O(n^2)$ in time $O(n^3)$.* □

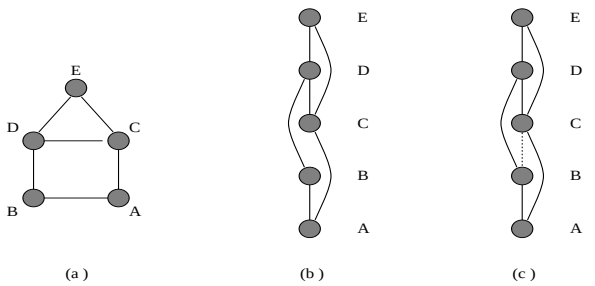


Figure 2: The interaction graph of φ_2

Corollary 2: *Given a directional extension $E_d(\varphi)$ of a 2-cnf theory φ, the entailment of any clause involving the first c symbols in d is $O(c^3)$.* □

4.1 Induced width

Let $\varphi = \varphi(Q_1, ..., Q_n)$ be a cnf formula defined on the variables $Q_1, ..., Q_n$. The *interaction graph* of φ, denoted $G(\varphi)$, is an undirected graph that contains one node for each propositional variable and an arc connecting any two nodes whose associated variables appear in the same clause. The interaction graph of φ_2 is given in Figure 2a. We can find a bound on the size of all theories having the same interaction graph using some properties of the graph.

Definition 1: Given a graph G and an ordering of its nodes d, the *parent set* of a node A relative to d is the set of nodes connected to A that precede A in the ordering d. The size of this parent set is the *width* of A relative to d. The width $w(d)$ of an ordering d is the maximum width of nodes along the ordering, and the width w of a graph is the minimal width of all its orderings [11, 7].

Lemma 1: *Given an interaction graph $G(\varphi)$ and an ordering d, if A is a node having $k-1$ parents, then there are no more than 3^k clauses in the bucket of A; if $w(d) = w$, then the size of the corresponding theory is $O(n \cdot 3^w)$.*

Proof: The bucket of A contains clauses defined on k literals only. For the set of $k-1$ symbols there are at most $\begin{pmatrix} k-1 \\ i \end{pmatrix}$ subsets of i symbols. Each subset can be associated

with at most 2^i clauses (i.e., each symbol can appear either positive or negative in a clause), and A can be also positive or negative. Therefore we can have at most

$$2 \cdot \sum_{i=0}^{k-1} \left(\begin{array}{c} k-1 \\ i \end{array} \right) 2^i = 2 \cdot 3^{k-1}. \quad (1)$$

clauses. Clearly, if the parent set is bounded by w, the extension is bounded by $O(n \cdot 3^w)$. □

Directional resolution applied along an ordering d to a theory having graph G adds new clauses and, accordingly, changes the interaction graph. The concept of induced graph will be defined to reflect those changes.

Definition 2: Given a graph G and an ordering d, the graph generated by recursively connecting the parents of G, in a reverse order of d, is called the *induced graph* of G w.r.t. d, and is denoted by $I_d(G)$. The width of $I_d(G)$ is denoted by $w^*(d)$ and is called the *induced width* of G w.r.t. d.

The graph in Figure 2a, for example, has width 2 along the ordering A, B, C, D, E (Figure 2b). Its induced graph is given in Figure 2c. The induced width of G equals 2.

Lemma 2: *Given a theory φ and an ordering d, the interaction graph $G(E_d(\varphi))$ of the directional extension of φ along d is a subgraph of $I_d(G(\varphi))$.*

Proof: The proof is done by induction on the variables along the ordering d. The induction hypothesis is that all the arcs incident to $Q_n, ..., Q_i$ in the $G(E_d(\varphi))$ appear also in $I_d(G(\varphi))$. The claim is true for Q_n, since its connectivity is the same in both graphs. Assume that the claim is true for $Q_n, ..., Q_i$ and we will show that it holds also for Q_{i-1}, namely, if (Q_{i-1}, Q_j), $j < i-1$, is an arc in $G(E_d(\varphi))$, then it also belongs to $I_d(G(\varphi))$. There are two cases: either Q_{i-1} and Q_j appeared in the same clause of the initial theory, φ, so they are connected in $G(\varphi)$ and, therefore, also in $I_d(G(\varphi))$, or a clause containing both symbols was added during directional resolution. In the second case, that clause was obtained while processing some bucket $Q_t, t > i-1$. Since Q_{i-1} and Q_j appeared in the bucket of Q_t, each must be connected to Q_t in $G(E_d(\varphi))$ and, by the induction hypothesis,

they will also be connected in $I_d(G(\varphi))$. Therefore, Q_{i-1} and Q_j would become connected in $I_d(G(\varphi))$, when connecting the parents of Q_t. □

Theorem 5: *Let $\varphi = \varphi(Q_1, ..., Q_n)$ be a cnf, $G(\varphi)$ its interaction graph, and $w^*(d)$ its induced width along d; then, the size of $E_d(\varphi)$ is $O(n \cdot 3^{w^*(d)})$ and the time complexity of directional resolution along the ordering d is $O(n \cdot 9^{w^*(d)})$.*

Proof: The result follows from lemmas 1 and 2: the interaction graph of $E_d(\varphi)$ is a subgraph of $I_d(G)$, and the size of theories having $I_d(G)$ as their interaction graph is bounded by $O(n \cdot 3^{w^*(d)})$. The time complexity of directional resolution is bounded by $O(n \cdot |bucket_i|^2)$, where $|bucket_i|$ is the size of the largest bucket (remember, that resolution on a bucket takes time quadratic in the bucket size). From lemma 1 we get $|bucket_i| = O(3^{w^*(d)})$, therefore the time complexity is $O(n \cdot 9^{w^*(d)})$. Note, that this deduction implicitly assumes that the algorithm eliminates duplicate clauses. □

As follows from the last theorem, theories with the bounded induced width would constitute a tractable class for directional resolution. It is known that the induced width of a graph embedded in a *k-tree* is bounded by k [1]. Here is a recursive definition of k-trees.

Definition 3: *(k-trees)*

1. A clique of size k (complete graph with k vertices) is a k-tree.
2. Given a k-tree defined on $Q_1, ..., Q_{i-1}$, a k-tree on $Q_1, ..., Q_i$ can be generated by selecting a clique of size k and connecting Q_i to every node in that clique.

Corollary 3: *If φ is a formula whose interaction graph can be embedded in a k-tree then there is an ordering d such that the time complexity of directional resolution on that ordering is $O(n \cdot 9^k)$.* □

Finding an ordering yielding the smallest induced width of a graph is NP-hard [1]. However, any ordering d yields an easily computed bound, $w^*(d)$. Consequently, when given a theory and its interaction graph, we will try to find an ordering that yields the smallest

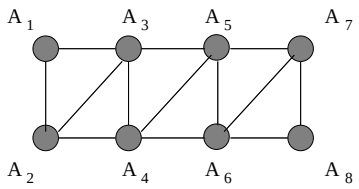


Figure 3: The interaction graph of φ_8 in example 3:$\varphi_8 = \{(A_1, A_2, \neg A_3), (\neg A_2, A_4), (\neg A_2, A_3, \neg A_4), (A_3, A_4, \neg A_5), (\neg A_4, A_6), (\neg A_4, A_5, \neg A_6), (A_5, A_6, \neg A_7), (\neg A_6, A_8), (\neg A_6, A_7, \neg A_8)\}$

width possible. Several heuristic orderings are available (see [2]). Important special tractable classes are those having $w^* = 1$ (namely, the interaction graph is a tree) and those having $w^* = 2$, called series parallel networks. These classes can be recognized in linear time. In general, given any k, graphs having induced width of k or less can be recognized in $O(exp(k))$.

Example 3: Consider a theory φ_n over the alphabet $\{A_1, A_2, , ..., A_n\}$. The theory φ_n has a set of clauses indexed by i, where a clause for i odd is given by $(A_i, A_{i+1}, \neg A_{i+2})$ and two clauses for i even are given by $(\neg A_i, A_{i+2})$ and $(\neg A_i,\ A_{i+1},\ \neg A_{i+2})$. The reader can check that the induced width of such theories along the natural order is 2 and thus the size of the directional extension will not exceed $18 \cdot n$ (see lemma 1). For given ordering of variables the induced graph is identical to the original graph (see figure 3).

4.2 Diversity

The concept of induced width frequently leads to a loose upper bound on the number of clauses recorded by directional resolution. In example 3, for instance, only 6 clauses were generated by directional-resolution when processed in the given order, even without eliminating subsumption and tautologies in each bucket, while the computed bound is $18 \cdot 8 = 144$. The induced graph may not be a tight bound for the interaction graph of $E_d(\varphi)$. Consider, for instance, the two clauses $(\neg A, B), (\neg C, B)$ and the order $d = A, C, B$. When bucket B is processed, no clause is added because B is positive in both clauses, nevertheless, nodes A and C will be connected in the induced graph. In this subsection, we introduce a more

refined parameter, called *diversity*, based on the observation that a propositional letter can be resolved upon only when it appears both positively and negatively in different clauses. Extensions to this parameter will later attempt at bounding the number of resolvents in a bucket.

Definition 4: (diversity of a theory)
Given a theory φ and an ordering d, let Q_i^+ (Q_i^-) denote the number of times Q_i appears positively (negatively) in $bucket_i$ w.r.t. d. The diversity of Q_i relative to d, $div(Q_i)$, is $Q_i^+ \times Q_i^-$. The *diversity of an ordering* d, $div(d)$, is the maximum diversity of its variables w.r.t. the ordering d and the *diversity of a theory*, div, is the minimal diversity over all its orderings.

Theorem 6: *Algorithm* min-diversity *(Figure 4) generates a minimal diversity ordering of a theory.*

Proof: Let d 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 and if pushed down, it must be replaced by a variable whose diversity is either equal to or higher than the diversity of Q_i. □

Theorem 7: *The complexity of algorithm min-diversity is $O(n^2 \cdot c)$, where c is the number of clauses in the input theory.*

Proof: Computing the diversity of a variable takes $O(c)$ time, and the algorithm checks at most n variables in order to select one with the smallest diversity at each of n steps. This yields the total $O(n^2 \cdot c)$ complexity. □

4.2.1 Discovering causal structures

The concept of diversity yields new tractable classes. If d is an ordering having a zero diversity, directional resolution will add no clauses to φ along d. Namely,

min-diversity (φ)

1. For $i = n$ to 1 do

2. 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 4: Algorithm *min-diversity*

Theorem 8: *Theories having zero diversity are tractable and can be recognized in linear time.* □

Example 4: Let $\varphi = \{(G, E, \neg F), (G, \neg E, D), (\neg A, F), (A, \neg E) (\neg B, C, \neg E) (B, C, D)\}$. The reader can verify that the ordering $d = A, B, C, D, E, F, G$ is a zero-diversity ordering of φ.

Note that the diversity of theories in example 3 along the specified ordering, is 1.

Zero-diversity theories generalize the notion of causal theories defined for general networks of multivalued relations [8]. According to the definition in [8], theories specified in the form of cnfs would correspond to *causal* if there is an ordering of the symbols such that each bucket contains a single clause, and consequently has zero diversity. Note that even when a general theory is not zero-diversity it is better to put zero-diversity literals last in the ordering (so that they will be processed first). Then, the size of the directional-extension is exponentially bounded in the number of literals having only strictly-positive diversities. In general, however, the parameter of interest is the diversity of the directional extension $E_d(\varphi)$ rather than the diversity of φ.

Definition 5: (induced diversity)
The *induced diversity of an ordering* d, $div^*(d)$, is the diversity of $E_d(\varphi)$ along d, and the *induced diversity of a theory*, div^*, is the minimal induced diversity over all its orderings.

Since $div^*(d)$ bounds the *added* clauses generated from each bucket, we can trivially bound the size of $E_d(\varphi)$ using div^*: for every d, $|E_d(\varphi)| \leq |\varphi| + n \cdot div^*(d)$. The problem is

that even for a given ordering d, $div^*(d)$ is not polynomially computable, and therefore, the bound is not useful. It can, however, be used for some special cases for which the bound is precomputable. We will use two principles to bound div^*.

1. Identify cases where $div = div^*$ can be polynomially recognizable.

2. Identify cases where all the added resolvents are superfluous, giving the notion of zero-diversity a broader interpretation.

For most theories and most orderings $div^*(d) > div(d)$. A special counter example we observed are the zero-diversity theories for which $div^*(d) = div(d) = 0$. We next identify a subclass of diversity-1 theories whose div^* remains 1.

Theorem 9: *A theory $\varphi = \varphi(Q_1, ..., Q_n)$, has $div^* \leq 1$ and is therefore tractable, if each symbol Q_i satisfies one of the following conditions: a. it appears only negatively; b. it appears only positively; c. it appears in exactly two clauses.*

Proof: The proof is by induction on the number of symbols. Clearly, the diversity of the top literal is at most 1. If it is of zero diversity, no clause is added during processing; if it is of diversity 1, then at most one clause is added. Assume it is added to bucket Q_j. If Q_j is a single-sign symbol, it will remain so, namely, the diversity of its bucket will be zero. Else, since there are at most two clauses containing Q_j, and one of these was at 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 we can show that processing Q_{i-1} will leave the diversity at most 1. The argument is identical to the base case of the induction. □

The set of theories in example 3 has $div^* = 1$. Note though, that we can easily create examples with high w^* having $div^* \leq 1$.

We next extend the class of causal theories by attributing a wider meaning to "zero" diversity. For the class of zero diversity theories defined earlier directional resolution will not generate any new clauses. Sometime, however, directional resolution generate clauses which are tautologies, or which are all subsumed in the original set of clauses.

Example 5: The odd-parity relation over n propositional symbols $P_1, ..., P_n$ allows all models with an odd number of "true" assignments. This theory has a nice representation using an additional set of n symbols $Q_1, ..., Q_n$. Symbol Q_i denotes the parity of the first i propositions $P_1, ..., P_i$. The general definition of the theory is given by:

$$P_1 \rightarrow Q_1$$

$$\neg P_1 \rightarrow \neg Q_1$$

The i^{th} definitions is

$$P_i, Q_{i-1} \rightarrow \neg Q_i$$

$$P_i, \neg Q_{i-1} \rightarrow Q_i$$

$$\neg P_i, Q_{i-1} \rightarrow Q_i$$

$$\neg P_i, \neg Q_{i-1} \rightarrow \neg Q_i$$

If we want to state that the relation is odd-parity we add the proposition, Q_n. Consider now the ordering: $d = (P_1, Q_1, P_2, Q_2, ..., P_n, Q_n)$. Clearly this ordering has diversity equal 2. Nevertheless its diversity equals to its induced diversity if we eliminate tautologies in resolvents. This leads to the following definition:

Definition 6: [Extended zero diversity]
Given a theory φ and an ordering d, Q_i is said to be of *extended zero diversity* iff all the resolvents generated when $bucket_i$ is processed are either tautologies, or they are subsumed by clauses in $bucket_1, ..., bucket_{i-1}$. An ordering has extended zero diversity iff all its buckets have extended zero diversity. A theory has extended zero diversity iff there exists an extended-zero-diversity ordering of its variables.

Theorem 10: *Algorithm* recognize-extended-zero-div *(see Figure 5) is guaranteed to recognize extended zero diversity.*

recognize-extended-zero-div(φ)

1. For $i = n$ to 1 do

2. Find a letter Q such that all its resolvents over the theory $\varphi_i = \varphi - \cup_{j=i+1}^{n} bucket_j$ are subsumed by φ_i. If no such letter exists declare failure. Else put Q as i^{th} in the ordering.

Figure 5: Algorithm recognizing extended zero diversity

Proof: Let S denote the set of variables of a theory φ. Assume that there exists an extended-zero-diversity ordering d of variables in S, but the algorithm could not find any such ordering. In other words, at some step i, after $Q_n, ..., Q_{i+1}$ were selected, the algorithm was not able to find any extended-zero-diversity variable among $S_i = S - \cup_{j=i+1}^{n} Q_j$. Now, let Q be the highest variable in S w.r.t. d. Then any resolvent obtained in Q's bucket is subsumed by clauses in lower buckets of d. Those buckets correspond to the variables in $S_i - \{Q\}$ and, possible, some of $Q_n, ..., Q_{i+1}$. Now we delete all the clauses containing $Q_n, ..., Q_{i+1}$ from those buckets, and the resulting theory will be exactly φ_i. But then then all resolvents in Q's bucket are subsumed in the buckets of $S_i - \{Q\}$, not in any one of $Q_n, ..., Q_{i+1}$ (subsumed clause should include all the variables of the subsuming clause), i.e. subsumed by φ_i. This contradicts our assumption. □

Theorem 11: *The complexity of the algorithm* recognize zero-diversity *is* $O(n^2 \cdot |\varphi|^3)$, *where* $|\varphi|$ *is the number of clauses in* φ.

Proof: For each variable Q at the i-th step of the algorithm $O(|\varphi|^2)$ clauses are generated by resolving over this variable, and each clause is checked for subsumption in $O(|\varphi|)$ time, which yields $O(|\varphi|^3)$ complexity of checking a variable for extended zero diversity. The algorithm performs n steps, checking no more than n variables at each step, i.e. the total bound on algorithm's complexity is $O(n^2 \cdot |\varphi|^3)$. □

DP-backtracking(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. $\quad Q$ = some unassigned variable;
6. $\quad$ return(DP-backtracking($\varphi \wedge Q$) $\vee$
   $\quad\quad$ DP-backtracking($\varphi \wedge \neg Q$))

Figure 6: Davis-Putnam procedure

5 Bounded directional resolution

Since the algorithm directional resolution is time and space exponential in the worst case, we propose an approximate algorithm called *bounded directional resolution* (BDR). The algorithm records clauses of size k or less when k is a constant. Consequently, its complexity is polynomial in k. Algorithm bounded directional resolution parallels algorithms for directional k-consistency in constraint satisfaction problems [7].

6 Experimental evaluation

In this section we report experimental results demonstrating advantages and drawbacks of the algorithms directional resolution (DR), bounded directional resolution (BDR), and DP-backtracking on problems with different structures. A combination of the last two algorithms called BDR-DP is proposed as an overall most efficient algorithm among them.

Directional resolution has been implemented in accordance with the algorithm described in section 3. DP-backtracking is a version of the Davis-Putnam procedure (see Figure 6)

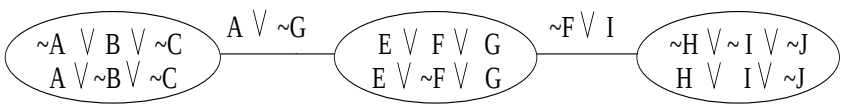


Figure 7: An example of a theory with the chain structure

that uses a loop construction instead of recursion in order to increase space efficiency. The algorithm has been also augmented with the *2-literal clause heuristic* proposed in [4]. The heuristic suggests to instantiate next a variable that would cause the largest number of unit propagations. The number of possible unit propagations is approximated by the number of 2-literal clauses in which the variable appears. The modified version significantly outperforms DP-backtracking without this heuristic [4]. In order to find a solution, DR was followed by DP-backtracking without the 2-literal clause heuristic so that the order of variables was fixed. As the theory dictates, no deadends occur when DP-backtracking runs after DR on the same ordering, and the time it takes is linear in the size of DR's output theory. Algorithm BDR, since it is incomplete for satisfiability, was followed by DP-backtracking augmented with the 2-literal clause heuristic. We call this combination BDR-DP.

We tested the algorithms on different orderings of variables: *input ordering* as used by the random problem generator, *min-width ordering*, and *min-diversity ordering*. Given an interaction graph, min-width ordering selects a variable with the smallest degree, and puts it last in the ordering; the node is eliminated from the graph and the ordering continues recursively. Min-diversity ordering was described earlier.

In order to test the algorithms on problems with different structures several random generators were used. *Uniform k-cnfs* were obtained using the generator proposed in [17]. It takes as an input the number of variables n, the number of clauses m, and the number of literals per clause k, and obtains each clause by choosing k variables randomly from the set of n variables and by determining the polarity of each literal with probability $p = 0.5$. Different values of p were also used. We did not check clause uniqueness since for

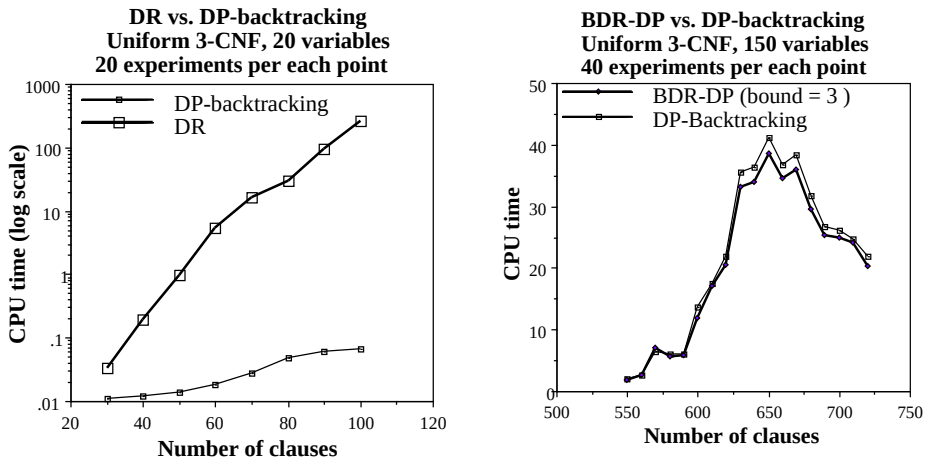
large number of variables it is unlikely that identical clauses will be generated. Our second generator, called *mixed cnfs*, produces theories containing clauses of length k_1 or k_2. The third generator, called *chains*, obtains a sequence of n independent uniform k-cnf theories (called *subtheories*), and connects them in a chain by $(n-1)$ 2-cnf clauses. Each 2-cnf clause contains variables from two consecutive subtheories in the chain (see Figure 7). Similarly, we connected sequences of independent theories into a tree structure.

Both chains and tree-structures described above belong to a class of random *embeddings in k-trees* [1]. We implemented a generator, called (k,m)-*trees*, which generalizes the idea of k-trees. A (k,m)-tree is a tree of cliques, each having $(k+m)$ nodes, where k is the size of intersection between each two neighbouring cliques. Therefore, conventional k-trees are $(k,1)$-trees according to our definition. The (k,m)-tree generators takes as an input k, m, the number of cliques in a (k,m)-tree *Ncliques*, and the number of clauses to be added per each new clique, *Nclauses*. It generates the first clique of size $k+m$ with *Nclauses* clauses in it, then, until *Ncliques* are generated, it chooses randomly a previously generated clique, randomly selects k variables out of this clique, adds m new variables and generates *Nclauses* clauses on the new clique. We should notice that *Nclauses* is *not* the number of clauses per each clique, because each new clique shares some variables with the previously (and, possible, future) generated cliques. The induced width of a (k,m)-trees is bounded by $k+m-1$.

6.1 Results for Problems with Uniform Structure

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

We compared DP-backtracking with DR on uniform k-cnfs for k=3,4,5 and on mixed theories. In all these cases DP-backtracking significantly outperforms DR. It was observed that the complexity of DR indeed grows exponentially with the size of problems (see Figure



(a) DR and DP-backtracking (b) BDR-DP and DP-backtracking

Figure 8: DP-backtracking, DR and BDR on uniform 3-cnfs

8a). We show the results for 3-cnfs with 20 variables only. On larger problems DR often ran out of memory due to the large number of generated clauses.

Since DR was so inefficient for solving uniform k-cnfs we used Bounded Directional Resolution (BDR) followed by DP-backtracking (BDR-DP) using different bounds. Our experiments show that BDR generates almost no new clauses when running on uniform k-cnf theories with a bound k or less. On the other hand, when the bound is strictly greater than k, running just BDR takes too long. The only promising case occurs when the bound equals k. In this case relatively few clauses were added by BDR which therefore ran much faster. DP-backtracking often ran a little faster on the BDR's output theory than on the original theory, and, therefore, the combined algorithm was somewhat more efficient than DP-backtracking (see Figure 8b). Our experiments have shown that on larger problems BDR-DP becomes more efficient relatively to DP-backtracking.

BDR-DP with bound 3 was also more efficient than DP-backtracking on random 3-cnfs when varying the probability of a literal to appear positive in a clause. The results of running BDR with bounds 3 and 4 on random cnfs with $p = 0.7$ are shown in Table 1.

Again, as in the case when $p = 0.5$, we can see that BDR-DP with bound 3 is almost

Table 1: DP-backtracking (DPB) versus BDR-DP with bounds 3 and 4 on uniform 3-cnfs

200 variables										
Probability of a literal to appear positive = 0.7										
Mean values on 20 experiments per each row										
Num of cls	**DPB:**		**BDR-DP (bound 3)**				**BDR-DP (bound 4)**			
	Time	Dead ends	BDR time	DP time	Dead ends	Number of new cls	BDR time	DP time	Dead ends	Number of 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

always more efficient that DP-Backtracking. Running BDR with larger bounds does not look promising, because even for bound 4 the preprocessing phase takes too long.

6.2 Results on chains

The behaviour of the algorithms on chains (and k-tree embeddings in general) differs dramatically from that on uniform instances. We found here extremely hard instances for DP-backtracking, orders of magnitude harder than those generated by the uniform model. In Table 2 we compare the performance of DP-backtracking on uniform 3-cnf problems and on 3-cnf chain problems with the same number of clauses. The problems contain 25 subtheories with 5 variables and 9 to 23 3-cnf clauses per subtheory, as well as 24 2-cnf clauses connecting subtheories in the chain. The corresponding uniform 3-cnf problems have 125 variables and 249 to 599 clauses. We tested DP-backtracking on both classes of problems.

Table 2 shows the mean values on 20 experiments where the number of experiments is per a constant problem size, i.e. per each row in the table. The min-diversity ordering have been used for each instance.

Table 2: DP-backtracking on uniform 3-cnfs and on chain problems of the same size

3-cnfs theories with 125 variables						
Mean values on 20 experiments per each row						
Num of clauses	**Uniform 3-cnfs**			**3-cnf chains**		
	% Sat	Time 1st solution	Dead ends	% Sat	Time 1st solution	Dead ends
249	100	0.2	0	100	0.3	0
299	100	0.2	0	100	0.4	1
349	100	0.2	3	70	9945.7	908861
399	100	0.2	2	25	2551.1	207896
449	100	0.4	17	15	185.2	13248
499	95	3.7	244	0	2.4	160
549	35	8.5	535	0	0.9	9
599	0	6.6	382	0	0.1	6

Most of the extremely hard chain problems with many deadends were found around the *cross-over point*, where about 50% of generated chain problems were satisfiable. As it was shown for uniform 3-cnfs [17, 4], the percentage of satisfiable problems and their complexity depend on the clauses/variables ratio. Small values of that ratio correspond to underconstrained problems most of which have many solutions and are easily solved by DP-backtracking. When the ratio is large, the problems become overconstrained, and mostly unsatisfiable. On the average, overconstrained problems are not very hard for DP-

Table 3: DR and DP-backtracking on 3-*cnf* chains

3-*cnf* chains with 125 variables: 25 subtheories, 5 variables in each								
Mean values on 20 experiments per each row								
Num of cls	SAT: %	**DP-backtracking**		**DR**				
		Time: solution	Dead ends	Time: 1st SAT only	Time: 1st solution	Number of new clauses	Size of max clause	Induced width
249	100	0.3	0	0.6	0.3	61	4.1	5.1
299	100	0.4	1	1.4	0.3	105	4.1	5.3
349	70	9945.7	908861	2.2	0.3	131	4.0	5.3
399	25	2551.1	207896	2.8	0.2	131	4.0	5.3
449	15	185.2	13248	3.7	0.3	135	4.0	5.5
499	0	2.4	160	3.8	0.0	116	3.9	5.4
549	0	0.9	9	4.0	0.0	99	3.9	5.2
599	0	0.1	6	4.6	0.0	93	3.6	5.2

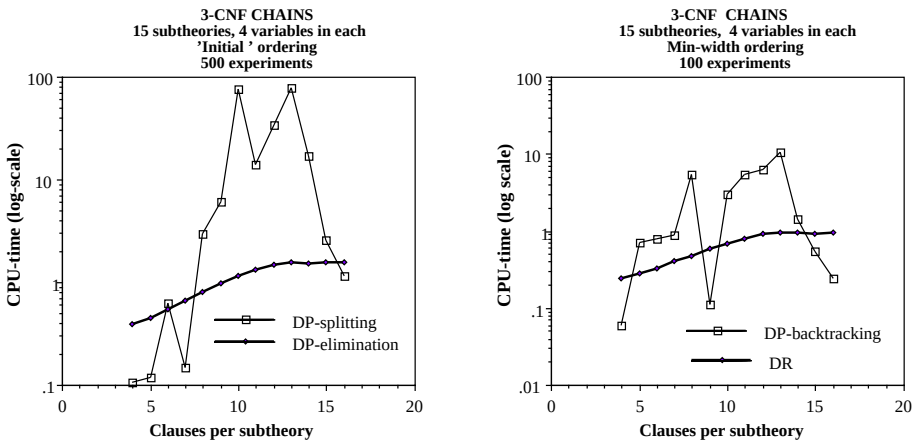
backtracking, which detects inconsistency early in the search. Most of the hard problems appear around the so called cross-over point, the transition point from mostly satisfiable to mostly unsatisfiable problems. According to experimental studies, for uniform 3-cnfs the transition occurs at the clauses/variables ratio approximately equal to 4.3. We observed that the crossover point for chains shifted towards a smaller ratio(see Table 2). Many chain problems around the crossover point were orders of magnitude harder for DP-backtracking than uniform 3-cnf problems of the same size, and also harder than average uniform 3-cnf problems at their crossover point. Directional resolution, on the other hand, behaved in a tamed way on chains and was sometimes more than 1000 times faster than DP-backtracking. In Table 3 we compare DP-backtracking with DR on the same chain problems as in Table 2. A detailed illustration in Table 4 lists the results on selected hard instances (number of

Table 4: DR and DP-backtracking on hard chain instances (number of deadends > 5000)

3-*cnf* chains with 125 variables: 25 subtheories, 5 variables in each				
Num of cls	SAT: 0 or 1	**DP-backtracking** Time: 1st solution	**DP-backtracking** Dead ends	**DR** Time: 1st solution
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

deadends exceeds 5000) from Table 3. We see that extremely hard instances (more than 100000 deadends), although rare, contribute most to the mean values.

All the experiments reported so far used min-diversity ordering. When experimenting with different orderings (input ordering and min-width ordering) we observed similar results (Figure 9). We also ran a small set of experiments with the actual code of *tableau* [4], that implements the Davis-Putnam procedure with various heuristics. Its behaviour on chain problems was similar to the one of our implementation. Some problem instances that were hard for our version of DP-backtracking were easy for tableau, however there was a subset of instances that were extremely difficult for both algorithms.



(a) input ordering (b) min-width ordering

Figure 9: DR and DP-Backtracking on chains with different orderings

Almost all the hard chain problems (for DP-backtracking) were unsatisfiable. One plausible explanation is the existence of an unsatisfiable subtheory that is last in the ordering. If all other subtheories are satisfiable, then DP-backtracking will try to re-instantiate variables from the satisfiable subtheories each time it encounters a deadend. Figure 10 shows an example of a chain of satisfiable theories where an unsatisfiable one appears almost at the very end of ordering. Min-diversity and min-width orderings do not guarantee that we avoid a situation like that. Not knowing the structure hurts DP-backtracking. Choosing the right ordering would help but this may be hard to recognize without some preprocessing. Other variants of backtracking that are capable of exploiting the structure like *backjumping*[13, 9] would avoid such useless re-instantiation of variables. Experiments with backjumping on the instances in Table 3 confirmed that it outperforms DP-backtracking by far (Figure 11). When experimenting with BDR-DP on chains we observed that all chain instances hard for DP-backtracking have been solved easily by BDR-DP. The performance of BDR-DP on chains was comparable to DR.

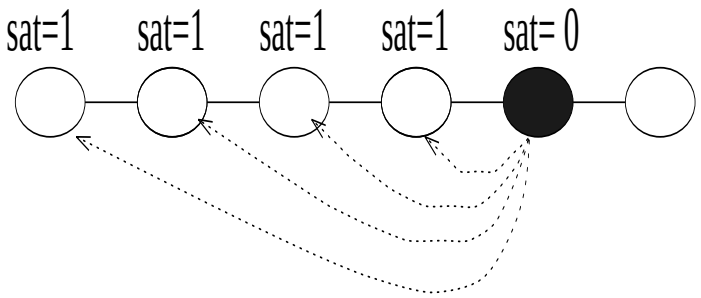


Figure 10: Illustration of a "hard chain problem"

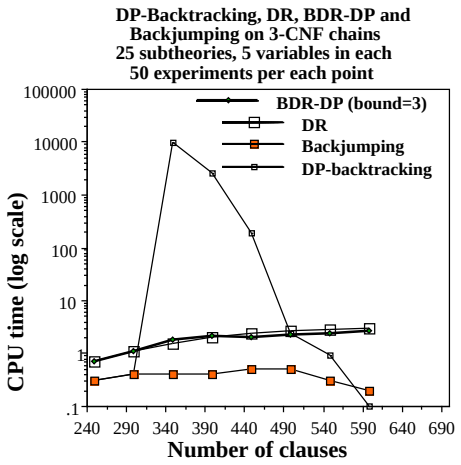


Figure 11: DP-Backtracking, DR and Backjumping on chains

6.3 Results on k-m-trees

The behaviour of both DP-backtracking and DR on (k, m)-trees is similar to the one we have observed on chains. Indeed, chains are $(2, m)$-trees. For fixed k, m and the number of cliques in a (k, m)-tree, we varied the number of clauses per clique and discovered, again, exceptionally hard problems for DP-backtracking around the (k, m)-trees' crossover point. Experiments on 1-4-trees and on 2-4-trees, with total of 100 cliques, show that DP-backtracking exceeded the limit of 20000 deadends (around 700 seconds) on 40% of 1-4-trees with $Nclauses = 13$, and on 20% of 2-4-trees with $Nclauses = 12$. Table 5 summarizes the results on (k, m)-trees. We terminate the algorithm once it reaches more than 20000 deadends. This explains why the difference in CPU time between BDR-DP and DP-backtracking is smaller than that on chains where we ran DP-backtracking until completion.

As in the case of chains, we observed that most of the exceptionally hard problems were unsatisfiable. The frequency of those hard instances and their hardness depend on the parameters of (k, m)-trees. For fixed m, when k is small, and the number of cliques is large, hard instances for DP-backtracking appear more often. The intuitive explanation is similar to one given for chains: we are likely to encounter a long sequence of almost independent (because the intersection size between cliques, k, is small) satisfiable subtheories which may end up with an unsatisfiable subtheory causing many deadends. If at the same time m is reasonably small than DR and BDR easily recognize the unsatisfiable subtheory.

Note that in one case a satisfiable instance that was relatively easy for DP-backtracking became very hard for the same algorithm after preprocessing by BDR. The problem is a 3-4-tree with 80 clusters and 9 clauses per cluster, solved by DP-backtracking in about 3 seconds with only 58 deadends. After preprocessing, DP-backtracking encountered 10000 deadends and was terminated without finding a solution after 227 seconds. It means that preprocessing (BDR) could hurt backtracking (DP) in some rare cases. Still, we concluded that BDR-DP is the overall superior among the algorithms considered in the paper: it is more efficient than DP-Backtracking and backjumping on uniform problems, although not significantly, and it recognizes easily unsatisfiable subproblems having huge amount of deadends if processed by

DP-backtracking.

6.4 Directional Resolution as Knowledge Compilation

Directional resolution may be viewed as a knowledge compilation algorithm. Our experiments show that answering queries on the directional extension of DR may be significantly faster than that on the original theory.

In order to determine if a clause is entailed by a theory, we add the negation of each literal in the clause to the theory and run DP-backtracking. In Table 6 we compare the time complexity of query answering for DP-backtracking before and after running DR, on a 3-cnf chain instance with 20 subtheories each containing 5 variables and 13 clauses. We terminate DP-backtracking if the number of deadends exceeds 50000. Deciding on satisfiability of that chain problem was also hard for DP-backtracking and took 360.6 seconds when DR solved it in just 0.6 seconds. Answering some queries was easy for DP-backtracking, but most of them required time comparable to that of deciding satisfiability. In general, query answering results on different random problem generators were similar to those for satisfiability checking. We observed that running DR as a preprocssing algorithm on chains is extremely useful, because on chains (and (k, m)-trees in general) the size of the directional extension is bounded, and query answering on the compiled theory is practically backtrack-free.

7 Related work and conclusions

Directional resolution belongs to a family of elimination algorithms first analyzed for optimization tasks in dynamic programming [2] and later used in constraint satisfaction [18, 7] and in belief networks [20]. The complexity of all those elimination algorithms is a function of the induced width w^* of the undirected graph characteristic of each problem instance. Although it is known that determining the w^* of an arbitrary graph is NP-hard, useful heuristics for bounding w^* are available.

Since propositional satisfiability is a special case of constraint satisfaction, the induced-

width could be obtained by mapping a propositional formula into the relational framework of a constraint satisfaction problem (see [3]), and then applying *adaptive consistency*, the elimination algorithm tailored for constraint satisfaction problems [7, 18]. We have recently shown, however, that this kind of pairwise elimination operation as performed by directional resolution is more effective. And, while it can be extended to any row-convex constraint problem [21] or to every 1-tight relations [22] it cannot decide consistency for arbitrary multi-valued networks of relations.

The paper makes three main contributions. First, we revive the old Davis-Putnam algorithm (herein called *directional resolution*) and mitigate the pessimistic analyses of DP-elimination by showing that the algorithm admits some known tractable classes for satisfiability and constraint satisfaction, including 2-*cnfs*, Horn clauses, causal networks, and bounded-width networks. Second, we identify new tractable classes based on the notion of *diversity* . Third, our empirical tests show that, while on uniform theories directional resolution is indeed ineffective, on problems with special structures, like *chains* and *k-trees*, having low w^*, directional resolution greatly outperforms DP-backtracking which is one of the most effective satisfiability algorithm known to date.

In conclusion, although directional resolution outperformsy DP-backtracking on some classes of problems, it is not advocated as an effective method for general satisfiability problems. Even when the structure is right, there are other structure-exploiting algorithms, like backjumping, that are likely to be more effective than BDR-DP in finding a satisfying solution. What we do advocate is that structure-based components should be integrated, together with other heuristics (like unit propagation), into any algorithm that tries to solve satisfiability effectively.

At the same time, we have shown that, for some structured domains, directional resolution is an effective knowledge compilation procedure, compiling knowledge into a form that facilitates efficient model generation and query processing.

Acknowledgements

We would like to thank Dan Frost, Eddie Schwalb and Rachel Ben-Eliyahu for comments on this paper. Also we would like to thank Dan Frost for running experiments with the backjumping algorithm.

References

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

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

[3] Ben-Eliyahu, R., and Dechter, R., Default Logic, Propositional Logic and Constraints, in *Proceedings of the National Conference on Artificial Intelligence* (AAAI-91), July 1991, Anaheim, CA, pp. 379-385.

[4] Crawford, J.M. and Auton, L.D., Experimental Results on the Cross-over Point in Satisfiability Problems, in *Proceedings of AAAI-93*, 1993, pp 21-27.

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

[6] Davis, M. and Putnam, H., A Computing Procedure for Quantification Theory, *J. ACM* 7:201-215 (1960).

[7] Dechter, R., and Pearl, J. Network-based Heuristics for Constraint Satisfaction Problems, *Artificial Intelligence* 34:1-38 (1987).

[8] Dechter, R., and Pearl, J., Directed Constraint Networks: A Relational Framework for Causal Models, in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (IJCAI-91), Sidney, Australia, August 1991, pp. 1164-1170.

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

[10] Even, S., Itai, A., and Shamir, A., On the Complexity of Timetable and Multi-Commodity Flow", *SIAM Journal on Computing*, 5:691-703 (1976).

[11] Freuder, E.C., A Sufficient Condition for Backtrack-Free Search, *J. ACM*, 29:24-32 (1982).

[12] Galil, Z., On the Complexity of Regular Resolution and the Davis-Putnam Procedure, *Theoretical Computer Science* 4:23-46 (1977).

[13] Gashnig, J., Performance Measurement and Analysis of Certain Search Algorithms, Technical Report CMU-CS-79-124, Carnegie Mellon University, 1979.

[14] Goerdt, A., Davis-Putnam Resolution Versus Unrestricted Resolution, *Annals of Mathematics and Artificial Intelligence*, 6:169-184 (1992).

[15] Goldberg, A., Purdom P., and Brown, C., Average Time Analysis of Simplified Davis-Putnam Procedures, *Information Processing Letters*, 15:72-75 (1982).

[16] McAllester, D., Private communication.

[17] Mitchell, D., Selman, B., and Levesque, H., Hard and Easy Distributions of SAT Problems, in *Proceedings of AAAI-92*, 1992, pp. 459-465.

[18] Seidel, R., A New Method for Solving Constraint Satisfaction Problems, in *Proceedings of the Seventh international joint conference on Artificial Intelligence* (IJCAI-81), Vancouver, Canada, August 1981, pp. 338-342.

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

[20] Lauritzen, S.L. and Spigelholter, D.J., Local Computations With Probabilities on Graphical Structures and Their Applications to Expert Systems, *Journal of the Royal Statistical Society, Series, B,* 5:65-74 (1988).

[21] van Beek, P. and Dechter, R., On the Minimality and Decomposability of Row-Convex Constraint Networks, *Journal of the Association of Computing Machinery (JACM)*, In press, 1995.

[22] van Beek, P. and Dechter, R., Constraint Tightness vs Global Consistency, November, 1994. Submitted manuscript.

Table 5: BDR-DP (bound 3) and DP-backtracking (termination at 20000 deadends) on (k, m)-trees

Mean values on 50 experiments per each row					
DP-backtracking		**BDR-DP with bound=3**			
Time: 1st solution	Dead ends	Time BDR only	Time: 1st solution DP after BDR	Dead ends DP after BDR	Number of new clauses
1-4-tree, Nclauses = 11, Ncliques = 100 Total: 401 variable, 1100 clauses					
233.2	7475	5.4	17.7	2	298
1-4-tree, Nclauses = 12, Ncliques = 100 Total: 401 variable, 1200 clauses					
352.5	10547	7.5	1.2	7	316
1-4-tree, Nclauses = 14, Ncliques = 100 Total: 401 variable, 1400 clauses					
1-4-tree, Nclauses = 13, Ncliques = 100 Total: 401 variable, 1300 clauses					
328.8	9182	9.8	0.25	3	339
1-4-tree, Nclauses = 14, Ncliques = 100 Total: 401 variable, 1400 clauses					
174.2	4551	11.9	0.0	0	329
2-4-tree, Nclauses = 12, Ncliques = 100 Total: 402 variable, 1200 clauses					
160.0	4633	6.0	1.6	25	341
2-4-tree, Nclauses = 13, Ncliques = 100 Total: 402 variable, 1300 clauses					
95.4	2589	8.3	0.1	0	390

Table 6: Query processing on a 3-cnf chain (20 subtheories, 5 variables in each)

20 queries of length 1			
DP before DR		**DP afterDR**	
Time	Deadends	Time	Deadends
0.1	13	0.0	1
0.0	1	0.0	1
224.2	30811	0.0	1
296.2	40841	0.0	1
240.5	33234	0.0	1
34.1	4753	0.0	1
34.0	4754	0.0	1
0.0	1	0.0	1
313.0	43324	0.0	1
362.0	50001	0.0	1
0.1	14	0.0	1
362.8	50001	0.0	1
361.2	50001	0.0	1
0.0	1	0.0	1
0.7	91	0.0	1
363.0	50001	0.0	1
68.1	9505	0.0	1
295.5	40842	0.0	1
0.0	1	0.0	1
367.4	50001	0.0	1