Chapter 10

Hybrids of Search and Inference: Time-Space tradeoffs

Throughout this book two primary constraint processing schemes emerge - those based on *conditioning* or search, and those based on *inference* or *derivation*. Search in constraint satisfaction takes the form of depth-first backtracking, while inference is performed by variable-elimination and tree-clustering algorithms, or by bounded local consistency enforcing. In analogy to human problem solving, conditioning corresponds to guessing (a value of a variable), or reasoning by assumption. The problem is divided into subproblems, conditioned on the instantiation of a subset of variables, each of which should be solved. On the other hand, inference correspond to thinking and understating the problem better. Inference-based algorithms derive and record new information, generating equivalent problem representation that facilitates easier solution.

Both search and inference algorithms have their advantages and disadvantages. Inference-based algorithms are efficient when the problem is sparse; when the induced-width of the problem is low, but otherwise require substantial time and memory. Search algorithms require exponential time in the worst case, but they need very little memory. Moreover, search often exhibit a much better average performance. Due to their complementary properties, the key to designing efficient algorithms is to combine inference-based and conditioning-based algorithms, and try to tailor the balance of inference and search to the problem instance. This may better utilize the benefit of each scheme and allow better performance guarantees, better space complexity and better overall average performance.

Indeed, due to its better average performance and minimal space requirements, backtracking is more popular than variable elimination for finding a single solution to constraint problems. Still, on problems with low width the latter techniques exploit the problems' structure, and they are can be more efficient than backtracking, as demonstrated empirically in Figure 10.1. The figure summarizes experiments over randomly

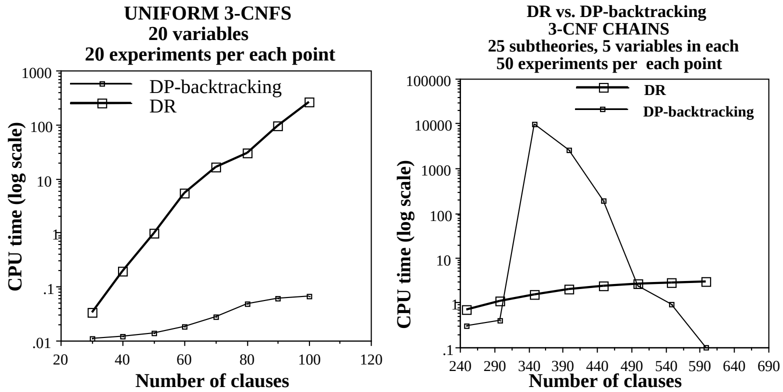


Figure 10.1: Average time performance of (a) DP-backtracking versus DR on uniform 3-cnfs; $w^* = 10$ to 18 (b) chain 3-cnfs, $w^* = 4$ to 7

generated satisfiability instances of two types, comparing DPLL backtracking for satisfiability with Directional Resolution (DR), the bucket-elimination algorithm for cnfs, presented in Chapter 8.

The figure shows that as expected, on uniform random 3-CNFs having large w^*, the complexity of the variable-elimination algorithm, DR, grew exponentially with the problem density while the performance of backtracking was much better. Even small problems having 20 variables already demonstrate the exponential behavior of DR (Figure 10.1a). On larger problems DR often ran out of memory. However, the behavior of the algorithms on chain problems having a small induced-width was completely different. Here, variable-elimination was by far more efficient than backtracking search.

An important property of elimination algorithms is that they are knowledge compilation algorithms. Instead of finding one solution, they compile a problem into another representation from which all solutions can be found in linear time.

The complementary characteristics of backtracking and elimination are summarized in Figure 10.2. There are several ways for combining search with inference, some of which have were hinted upon in previous chapters. One way is to use a restricted version of inference procedures as a preprocessing to search, yielding a restricted search space. Another approach is to alternate between both methods: apply conditioning to a subset of the variables, and then perform inference on the rest. One instantiation of this idea was explored in look-ahead backtracking algorithms. There, at *every* state of the search, namely, at each partial variable assignment, an approximate *bounded* inference algorithm (e.g., arc-consistency) is applied. This chapter presents two orthogonal architectures that can also be combined. The first is to reduce the general problem via conditioning to specific subproblems that are easy to solve for variable-elimination, and iterate via backtracking. The second starts with bounding the separators' sizes of tree-decompositions (see 9) and then

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

Figure 10.2: Comparison between backtracking and variable-elimination

applies conditioning search inside tree-clusterings having bounded separator sizes.

We will start with a special case of the first hybrid scheme called the cycle-cutset decomposition, move to this general hybrid framework, and demonstrate the methodology in the context of propositional satisfiability.

10.1 The Cycle Cutset Scheme

The hybrid algorithm presented in this section is based on identifying a *cycle cutset*, that is, a set of nodes that once removed, would render the constraint graph cycle-free, yielding a method called the cycle-cutset decomposition.

The method exploit the fact that variable instantiation changes the effective connectivity of the constraint graph. Consider a constraint problem whose graph is given in Figure 10.3a. For this problem, instantiating x_2 to some value, say a, renders the choices of values to x_1 and x_5 independent as if the pathway $x_1 - x_2 - x_5$ were blocked at x_2. Similarly, this instantiation blocks dependency in the pathway $x_1 - x_2 - x_4$, leaving only one path between any two variables. In other words, given that x_2 was assigned a specific value, the "effective" constraint graph for the rest of the variables is shown in Figure 10.3b, where the instantiated variable x_2 and its incident arcs are first deleted from the graph, and subsequently x_2 is duplicated for each of its neighbors. The constraint problem whose graph is shown in Figure 10.3(a) when $x_2 = a$ is identical to the constraint problem having the graph in 10.3(b) with the same assignment $x_2 = a$.

When the group of instantiated variables constitutes a cycle cutset, the resulting network is cycle free, (as shown in Figure 10.3b), and can be solved by the inference-

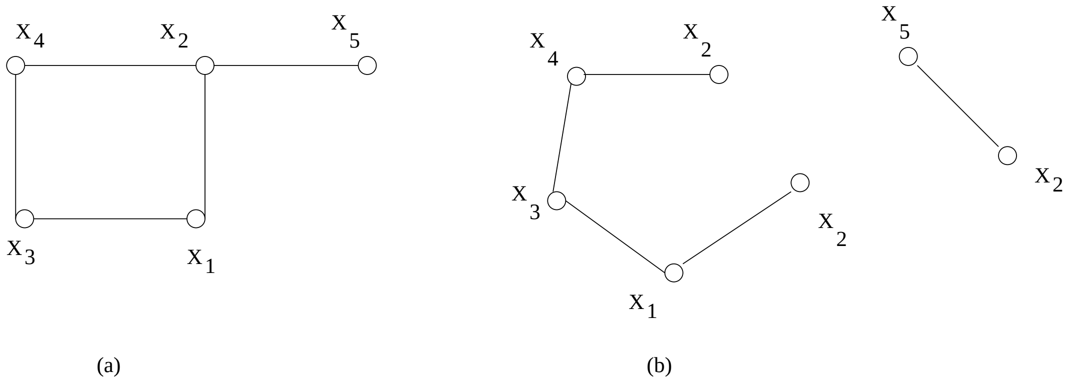


Figure 10.3: An instantiated variable cuts its own cycles.

based *tree-solving* algorithm. In most practical cases it would take more than a single variable to cut all the cycles in the graph. Thus a general way of solving a problem whose constraint graph contains cycles is to identify a subset of variables that cut all cycles in the graph, find a consistent instantiation of the variables in the cycle-cutset and then solve the remaining problem by the *tree algorithm*. If a solution to this restricted problem (conditioned on the cycle-cutset values) is found, then a solution to the entire problem is at hand. If not, another instantiation of the cycle cutset variables should be considered until a solution is found. If the task is to solve a constraint problem whose constraint graph is presented in Figure 10.3a, (assume x_2 has two values $\{a, b\}$ in its domain), first $x_2 = a$ must be assumed, and the remaining tree problem relative to this instantiation, is solved. If no solution is found, it is assumed that $x_2 = b$ and another attempt is made.

The number of times the tree-solving algorithm is invoked is bounded by the number of partial solutions to the cycle-cutset variables and therefore a small cycle-cutset is desirable. However, since finding a minimal-size cycle cutset can be shown to be computationally hard, it will be more practical to settle for a compromise and incorporate this scheme within backtracking search. Because *backtracking* works by progressively instantiating sets of variables, all that is necessary is to keep track of the connectivity status of the constraint graph. As soon as the set of instantiated variables constitutes a cycle cutset, the search algorithm is switched to the tree-solving algorithm on the restricted problem, i.e., either finding a consistent extension for the remaining variables (thus finding a solution to the entire problem) or concluding that no such extension exists (in which case backtracking takes place and another instantiation tried).

Some simple heuristic ordering of the variables that aim at finding a small cycle-cutset could be very beneficial here. One very simple heuristic is to order the variables in decreasing order of their degree in the constraint graph. Several greedy and approximation

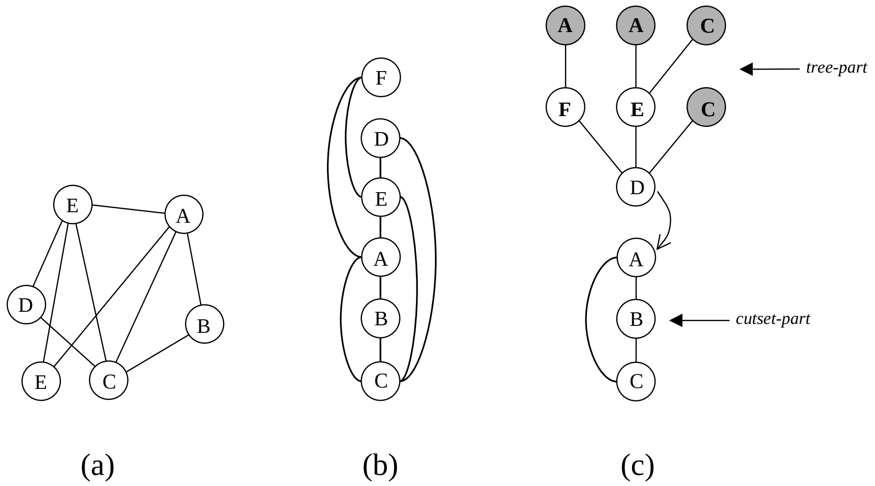


Figure 10.4: (a) a constraint graph (b) its ordered graph (c) The constraint graph of the cutset variable and the conditioned variable, where the assigned variables are darkened.

algorithms for this problem can be found in the literature (see chapter notes).

Example 10.1.1 Assume that backtracking instantiate the variables of the CSP represented in Figure 10.4a in the order C, B, A, E, D, F (Figure 10.4b). Backtracking will instantiate variables C, B and A, and then, realizing that these variables cut all cycles, will invoke a tree-solving routine on the rest of the problem: the tree-problem in Figure 10.4c with variables C, B and A assigned should be solved. If no solution is found, control returns to backtracking which will go back to variable A. □

Observe that the applicability of this idea is independent of the particular type of backtracking algorithm used over the cycle-cutset variables (e.g., naive backtracking, backjumping, backtrack with learning etc.). Also note that when the original problem has a tree-constraint graph, the cycle-cutset scheme coincides with a tree-algorithm, and when the constraint graph is complete, the algorithm reverts to regular backtracking.

Because in the worst-case all possible assignments to the cutset variables need to be tested, if c is the cycle-cutset size, k^c number of tree-structured subproblems need to be solved, each requiring $O((n-c)k^2)$ steps for the tree algorithm, yielding,

Theorem 10.1.2 *Algorithm cycle-cutset decomposition has time complexity of* $O(n \cdot k^{c+2})$ *where* n *is the number of variables,* c *is the cycle-cutset size and* k *is the domain size. The space complexity of the algorithm is linear.* □

10.2 Hybrids: conditioning first

The cycle-cutset scheme can be generalized. Rather than insisting on a conditioning subset (or a cutset) that cut all cycles, yielding width-1 subproblems, cutsets that create subproblems having bounded width that is higher than 1, may be allowed. This suggests a framework of hybrid algorithms parameterized by a bound b on the induced-width of subproblems solved by inference. The algorithm selects a set of *cutset* variables that when removed, the constraint graph has an induced-width bounded by b. We call such a conditioning set a *w-cutset*.

Definition 10.2.1 (w-cutset) *Given a graph G a subset of nodes is called a w-cutset iff when removed the resulting graph has an induced-width less or equal to w. A minimal w-cutset of a graph has a smallest size among all w-cutset of the graph. A cycle-cutset is a 1-cutset of a graph.*

Since conditioning on a value-assignment translates to nodes deletion in the constraint graph, the subproblems can be solved by variable-elimination in time and space exponential in b. In other words, the tree-algorithm in the cycle-cutset scheme can be generalized to variable-elimination. This yields algorithm, *elim-cond(b)* described in Figure 10.5. It runs backtracking search on the cutset variables and bucket-elimination (e.g., adaptive-consistency) on the remaining ones. The constraint problem $\mathcal{R} = (X, D, C)$ conditioned on an assignment $Y = \bar{y}$ and denoted by $\mathcal{R}_{\bar{y}}$, is $\mathcal{R}$ augmented with the unary constraints dictated by the assignment $\bar{y}$. Remember that when adaptive-consistency process a network $\mathcal{R}$ having a set of assignments $\bar{y}$ it may process the variables of assigned buckets first, and each is processed by the instantiation step (see Figure 8.2).

Given a bound b, elim-cond(b) allows bounding the amount of memory required while still maintaining desirable performance guarantee. In particular, b can be used to control the balance between search and variable-elimination, and thus affect the tradeoff between time and space.

Theorem 10.2.2 *Given $\mathcal{R} = (X, D, C)$, if elim-cond(b) is applied along ordering d when Y is a b-cutest of size c_b, then the space complexity of elim-cond(b) is bounded by $O(n \cdot exp(b))$ and its time complexity is bounded by $O(n \cdot exp(c_b + b))$,* $\Box$

Proof: Given a b-cutset assignment, the time and space complexity of the inference portion by elimination is bounded by $O(n \cdot k^b)$. Since in the worst-case, backtracking involves enumerating all possible instantiations of the b-cutset in $O(k^{c_b})$ time and linear space, the total time complexity is $O(n \cdot k^b \cdot k^{c_b}) = O(n \cdot k^{(b+c_b)})$. ∎

Theorem 10.2.2 calls for a new optimization task on graphs:

Algorithm elim-cond(b)
Input: A constraint network $\mathcal{R} = (X, D, C)$, $Y \subseteq X$ which is a b-cutset. d an ordering that starts with Y s.t. that induced-width along d of the conditional problem is bounded by b, $Z = X - Y$.
Output: A consistent assignment if there is one.

1. **while** $\bar{y} \leftarrow$ next partial solution of Y found by backtracking, **do**
   (a) $\bar{z} \leftarrow adaptive - consistency(\mathcal{R}_{Y=\bar{y}})$.
   (b) **if** $\bar{z}$ is not *false*, return solution=$(\bar{y}, \bar{z})$.
2. **endwhile**.
3. **return:** the problem has no solutions.

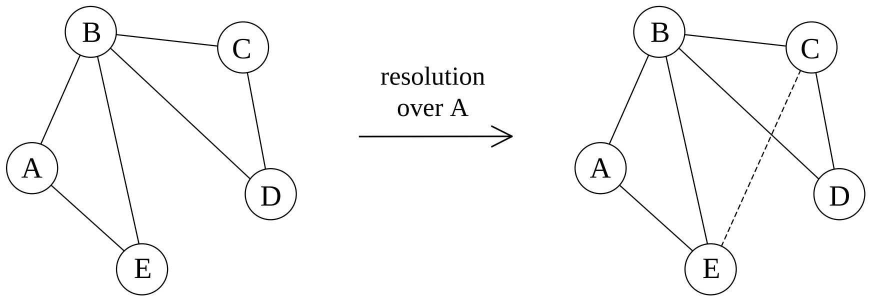
Figure 10.5: Algorithm *elim-cond(b)*

Definition 10.2.3 (finding a b-cutset) *Given a graph $G = (V, E)$ and a constant b, find a minimal b-cutset. Namely, find a smallest subset of nodes U, such that $G\prime = (V - U, E\prime)$, where $E\prime$ includes all the edges in E that are not incident to nodes in U, has induced-width less or equal b.*

Notice that verifying that a subset of nodes is a b-cutset is polynomial, although it is exponential in b. It can be accomplished by deleting the b-cutset from the graph and verifying that the remaining graph is bounded by b which is $O(exp(b))$. Moreover, given an ordering one case test if a conditioning set is a b-custet relative to the ordering, by inspecting the induced-width of the reduced ordered graph. Therefore, elim-cond(b) can be applied by using backtracking to all the variables in a given ordering until the problem conditioned on the partial assignment has an induced-width b or less *along the ordering*. Finally, conditioning and variable elimination can be interleaved dynamically as will be demonstrated in the next section.

In summary, the parameter b can be used to control the trade-off between search and inference. If $b \geq w_o^*$, where o is the ordering used by elim-cond(b), the algorithm coincides with adaptive-consistency. As b decreases, the algorithm requires less space and more time.

Indeed, there is no guaranteed worst-case time improvement over pure bucket-elimination. For example, it can be shown that the size of the smallest cycle-cutset (1-cutset), c_1 and



(a) (b)

Figure 10.6: (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.

the smallest induced width, w^*, obey the inequality $c_1 \geq w^* - 1$. Therefore, $1 + c_1 \geq w^*$, where the left side of this inequality is the exponent that determines the time complexity of elim-cond(b=1), while w^* governs the complexity of bucket-elimination. Alternatively we have

$$1 + c_1 \geq 2 + c_2 \geq ...b + c_b, ..., w^* + c_{w^*}$$

But since by definition $c_{w^*} = 0$ we get a hybrid scheme whose time complexity decrease as its space increases until it reaches the induced-width.

10.2.1 Hybrid algorithm for propositional theories

In this section we present a variant of elim-cond(b) for processing propositional cnfs. As noted in earlier chapters, the most celebrated backtracking search algorithm for propositional *cnf* theories is the backtracking DPLL algorithm that applies look-ahead using unit-propagation at each node. The bucket-elimination algorithm for cnfs is Directional Resolution (DR), which uses resolution as its variable-elimination operator, described in Chapter 8.

Remember that 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 10.6a. 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 10.6b.

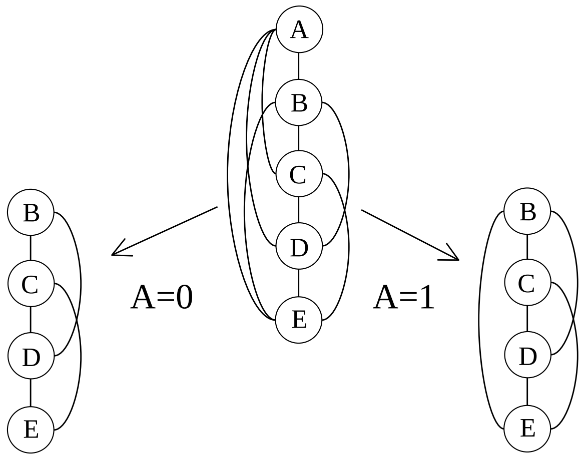


Figure 10.7: The effect of conditioning on A on the interaction graph of theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C \vee D), (\neg A \vee B \vee E \vee D), (\neg B \vee C \vee D)\}$.

The theory φ conditioned on the assignment $Y = \bar{y}$ is called a *conditional theory* of φ relative to $\bar{y}$, and is denoted by $\varphi_{\bar{y}}$. Since the effect of conditioning on a subset of propositions Y is deletion of variables in Y from the constraint graph, the *conditional graph* of φ with respect to $\bar{y}$, denoted $G(\varphi_{\bar{y}})$, is obtained by deleting the nodes in Y (and all their incident edges) from $G(\varphi)$. The *conditional induced width* of a theory φ relative to $\bar{y}$, denoted $w^*_{\bar{y}}$, is the induced width of the graph $G(\varphi_{\bar{y}})$.

Example 10.2.4 Figure 10.7 shows the interaction graph of theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C \vee D), (\neg A \vee B \vee E \vee D), (\neg B \vee C \vee D)\}$ along the ordering $o = (E, D, C, B, A)$ having induced width 4. Conditioning on A yields two conditional theories: $\varphi_{A=0} = \{(\neg C \vee E), (B \vee C \vee D), (\neg B \vee C \vee D)\}$, and $\varphi_{A=1} = \{(\neg C \vee E), (B \vee E \vee D), (\neg B \vee C \vee D)\}$. The ordered graphs of $\varphi_{A=0}$ and $\varphi_{A=1}$ are also shown in Figure 10.7. Clearly, $w^*_o(B) = 2$ for theory $\varphi_{A=0}$, and $w^*_o(B) = 3$ 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 $(\neg A \vee B \vee E \vee D)$ becomes satisfied. $\square$

Algorithm DP-DR(b), combines DPLL and DR, and uses also dynamic variable-ordering. The algorithm presented in Figure 10.9, takes as an input a propositional theory φ and a parameter b. If no inconsistency is discovered, DP-DR(b) 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 conditioned interaction graph, DP-DR(b) resolves upon Q (steps 4-9). Otherwise, it selects an unassigned variable (step 10), adds it to the b-cutset (step 11), and continues recursively with the conditional theory $\varphi \wedge \neg Q$.

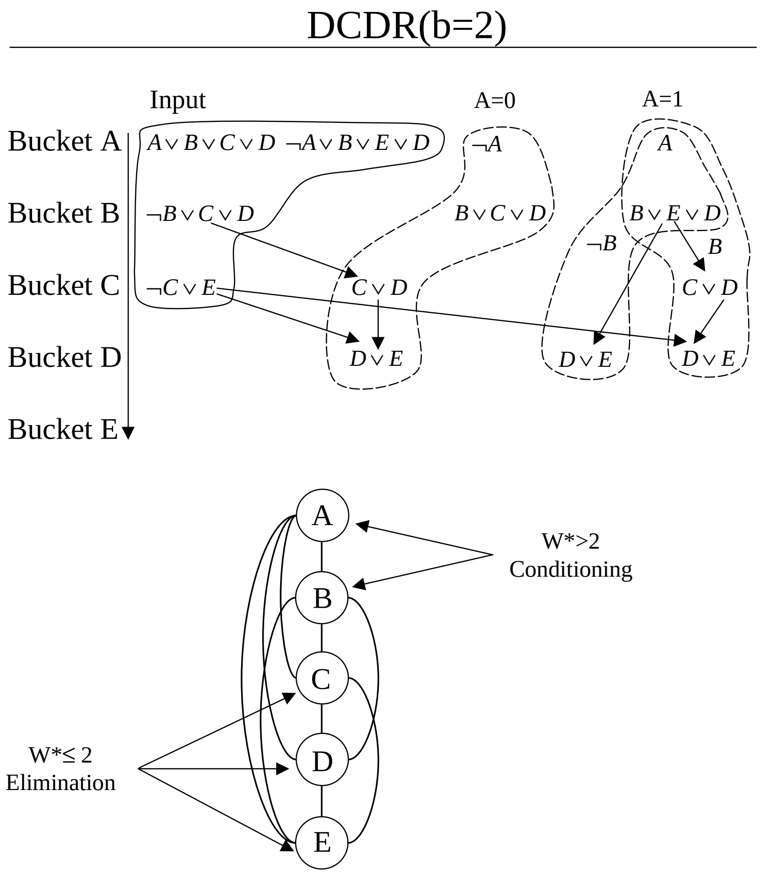


Figure 10.8: A trace of DP-DR(2) on the theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C \vee D), (\neg A \vee B \vee E \vee D), (\neg B \vee C \vee D)\}$.

An unassigned variable is selected using the same dynamic variable ordering heuristic that is used by DPLL. 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. Otherwise, it returns an assignment $\bar{y}$ to the cutset Y, and the conditional directional extension $E_o(\varphi_{\bar{y}})$ where o is the variable ordering dynamically constructed by the algorithm.

Algorithm DP-DR(2) is demonstrated in Figure 10.8 for the theory $\varphi = \{(\neg C \vee E), (A \vee B \vee C \vee D), (\neg A \vee B \vee E \vee D), (\neg B \vee C \vee D)\}$. 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

DP-DR(φ, b)
Input: A cnf theory φ over variables X; a bound b.
Output: A decision of whether φ is satisfiable. If it is, an assignment $\bar{y}$ to its conditioning variables, and the conditional directional extension $E_o(\varphi_{\bar{y}})$.
1. **if** unit_propagate(φ) = *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 conditioned 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. $Y \leftarrow Y \cup \{Q\}$;
12. return(DP-DR($\varphi \wedge \neg Q$, b) $\vee$
    DP-DR($\varphi \wedge Q$, b)).

Figure 10.9: Algorithm DP-DR(b).

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 10.8, 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 $A = 0$ (i.e., $\neg A$) yields a conditional theory $\{(\neg B \vee C \vee D), (B \vee C \vee D), (\neg C \vee E)\}$. Since the degrees of all the variables in the corresponding (conditional) interaction graph are now 2 or less (Figure 10.7a), 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 (Figure 10.7b), 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.

As a specialized version of elim-con(b), algorithm DP-DR(b) is guaranteed to generate clauses of length b or less by resolution. We can conclude,

Theorem 10.2.5 (complexity of DP-DR(b)) *The time complexity of algorithm DP-DR(b) is $O(n \cdot exp(b+c_b))$, where c_b is the largest cutset ever conditioned upon. The space complexity is $O(n \cdot exp(b))$.*

Empirical evaluation of DP-DR(b)

We conclude with showing results from typical experiments with different structured cnfs and its effect on $DP-DR$. The experiments focus on random uniform 3-cnfs having 100 variables and 400 clauses, as well as on $(4,5)$-trees with 40 cliques, 15 clauses per clique, and with $(4,8)$-trees, with 50 cliques, 20 clauses per clique, respectively. In general, a (k,m)-trees are trees of cliques each having $m+k$ nodes where and separators of size k. The 3-cnfs were generated randomly so that the theories graph will correspond to an (k,m)-tree. As expected, the performance of DP-DR(b) depends on the induced width of the theories.

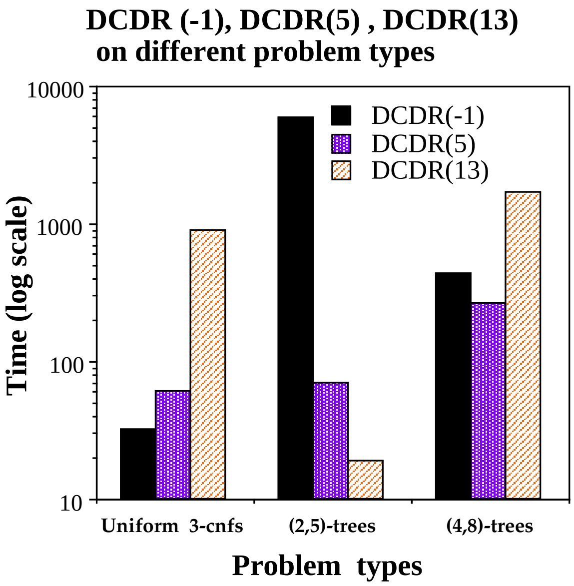


Figure 10.10: Relative performance of DP-DR(b) for $b = 0, 5, 13$ on different types of problems.

Figure 10.10 summarizes the results for DP-DR(0), DP-DR(5), and DP-DR(13) on the three classes of problems. The intermediate bound b=5 seems to be overall more cost-effective than both extremes, b= 0 and b=13. We see that time gain was achieved with intermediate values of b because backtracking average performance is better than its worst-case.

10.3 Hybrids: inference-first

In this section we will observe an orthogonal approach for a hybrid which is based on exploiting tree-decompositions as described in Chapter 9. We saw that the main drawback of algorithm cluster-tree elimination (CTE) which processes tree-decompositions is its memory needs. The space complexity of CTE is exponential in the largest separator size. In practice, this may be prohibitive, and therefore time-space tradeoff algorithms are necessary.

Algorithm CTE already hides a combination of variable elimination with search: while it computes functions on the separators using variable elimination and spends time and space exponential in the separator's size, the clusters themselves can be processed by enumeration or search.

The idea we present next is to trade even more space for time by allowing larger cliques but smaller separators. This can be achieved by combining adjacent nodes in a tree-decomposition that are linked by "fat separators", and keeping apart only nodes which are linked by bounded size separators. This process will increase the tree-width and hyper-width of the resulting tree. Consequently, as the size of the cliques grow, larger portions of the problem will be processed by search. The following paragraphs elaborate.

10.3.1 The super cluster tree elimination

Assume now that we have a problem whose tree-decomposition has tree-width r and separator size s, but that space restrictions do not allow the necessary $O(exp(s))$ memory required by CTE on such a tree. One way to overcome this problem is to collapse nodes in the tree connected by large separators to include the variables and constraints from both nodes. The resulting tree-decomposition has larger subproblems but smaller separators. This idea suggests a sequence of tree-decompositions and algorithms, parameterized by the sizes of their separators.

Definition 10.3.1 (Primary and secondary tree-decompositions) *Let T be a tree-decomposition of hypergraph $\mathcal{H}$. Let $s_0, s_1, ..., s_n$ be the sizes of the separators in T, listed in strictly descending order. With each separator size s_i we associate a tree decomposition T_i, generated by combining adjacent nodes whose separator sizes are strictly greater than s_i. $T = T_0$ is called the primary tree-decomposition, while T_i, when $i > 0$, is a secondary tree-decomposition. We denote by r_i the largest set of variables in any cluster of T_i and by hw_i the largest number of constraints in T_i.*

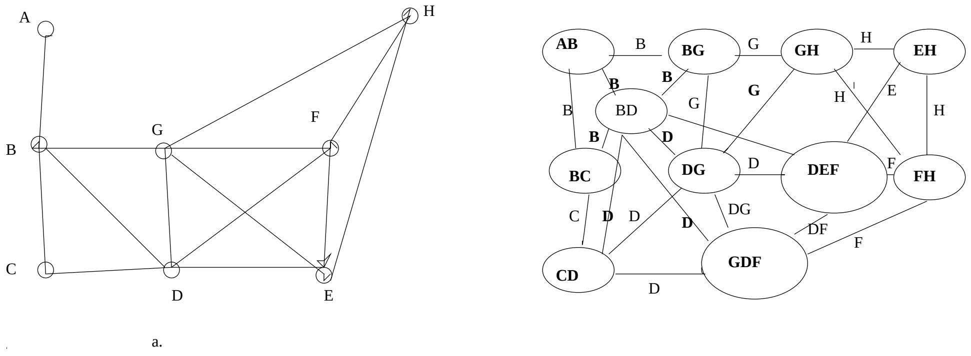


Figure 10.11: Primal (a) and dual (b) constraint graphs

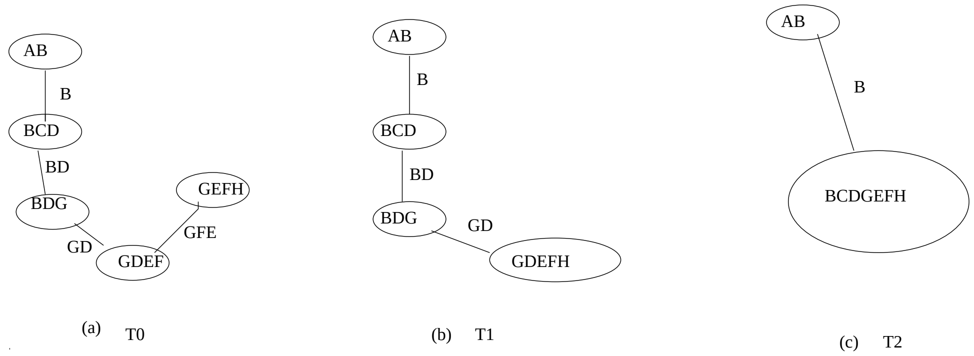


Figure 10.12: A tree-decomposition with separators equal to (a) 3, (b) 2, and (c) 1

Note that as s_i decreases, r_i and hw_i increase. Clearly, from Theorem 9.2.10 it follows that

Theorem 10.3.2 *Given a tree-decomposition T over n variables, having separator sizes $s_0, s_1, ..., s_t$ and corresponding secondary tree-decompositions having maximal number of nodes, $r_0, r_1, ..., r_t$, the complexity of CTE when applied to the secondary tree-decompositions has the following possible time and space bounds: $O(n \cdot exp(r_i))$ time, and $O(n \cdot exp(s_i))$ space (i ranges over all the of secondary tree-decomposition), respectively.*

Note again that each clique is processed by search. Namely, each solution created by backtracking, each of which is immediately projected on the separator and the projected solutions are accumulated. Any backtracking scheme can be used. We will call the algorithm SUPER-CLUSTER ELIMINATION (B), or $SCTE(b)$. It starts from a primary tree-decomposition, and then generates a tree-decomposition whose separator's size is bounded by b, which is subsequently processed by CTE. In the following example we assume a naive-backtracking search processing each clique.

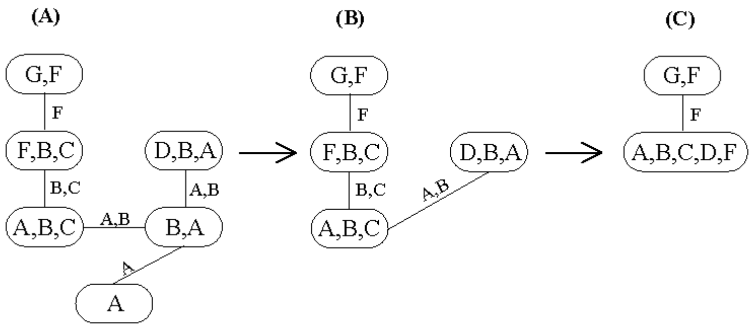


Figure 10.13: From a bucket-tree (left) to join-tree (middle) to a super-bucket-tree (right)

Example 10.3.3 Consider the constraint problem having the constraint graphs in Figure 10.11. The graph can be decomposed into the join-tree in Figure 10.12(a). If we allow only separators of size 2, we get the join tree T_1 in Figure 10.12(b). This structure suggests that we apply CTE in time which is exponential in the largest cluster, 5, while using space exponential in 2. If space considerations allow only singleton separators, we can use the secondary tree T_2 in Figure 10.12(c). We conclude that the problem can be solved either in $O(k^4)$ time (k being the maximum domain size) and $O(k^3)$ space using T_0, or in $O(k^5)$ time and $O(k^2)$ space using T_1, or in $O(k^7)$ time and $O(k)$ space using T_2. □

Superbuckets

Since bucket-elimination algorithms can be extended to bucket-trees, and since a bucket tree is a tree-decomposition, by merging adjacent buckets we generate a *super-bucket-tree* (SBT). This means that in the top-down phase of processing CTE, when applied to the super-bucket tree, several variables are eliminated at once. Note that one can always generate a join-tree from a bucket-tree by merging adjacent nodes. For illustration see Figure 10.13.

10.3.2 Decomposition into Non-separable Components

A special case of a tree-decomposition is when all the separators are required to be singleton variables. This type of tree-decompositions is attractive because it requires only linear space. While in general we cannot find the best tree-decompositions having a bounded separators' size in polynomial time, when the separators are singletons it is feasible. This is achievable using the graph notion of *non separable components* [55] discussed in this subsection.

Definition 10.3.4 (non-separable components) *A connected graph, $G = (V, E)$ is* *said to have a separation node v if there exist nodes a and b such that all paths connecting a and b pass through v. A graph that has a separation node is called separable, and one that has none is called non-separable. A subgraph with no separation nodes is called a non-separable component or a bi-connected component.*

An $O(\mid E \mid)$ algorithm exists for finding all the *non-separable components* and the *separation nodes*. It is based on a *depth-first search* traversal of the graph as discussed in Chapter 6. An important property of non-separable component is that that they are interconnected in a tree-structured manner [55].

Let G be a graph and SG be the *tree* whose nodes are the non-separable components $C_1, C_2, \ldots, C_r$ and the separating nodes are $V_1, V_2, \ldots, V_t$. Clearly the tree of non-separable components suggests a tree-decomposition where each node corresponds to a component, the variables of the nodes are those appearing in each component, and the constraints can be freely placed into a component that contains their scopes. Applying CTE to such a tree requires only linear space, but is time exponential in the components' sizes.

Example 10.3.5 Assume that the graph in Figure 10.14(a) represents a constraint network having unary, binary and ternary constraints as follows:
$\mathcal{R} = \{R_{AD}, R_{AB}, R_{DC}, R_{BC}, R_{GF}, D_G, D_F, R_{EHI}, R_{CFE}\}$. The non-separable components and their tree-structure are given in Figure 10.14(b,c). The ordering of components $d = (C_1, C_2, C_3, C_4)$ dictates super-clusters associated with variables $\{G, J, F\}$, $\{E, H, I\}$, $\{C, F, E\}$ and $\{A, B, C, D\}$. The initial partition into super-clusters and a schematic execution of CTE are displayed in Figure 10.15(a). We show the execution as a message passing along a superbucket tree. The messages sent from neighbors are placed inside the buckets. ρ_A^B is the message computed by bucket A and sent to bucket B. □

Theorem 10.3.6 (non-separable components) *Let $\mathcal{R} = (X, D, C)$ be a constraint network whose constraint graph has non-separable components of size r at the most, then the super-bucket elimination algorithm whose buckets are the non-separable components is time exponential $O(n \cdot exp(r))$ and linear in space.*

10.3.3 Hybrids of hybrids

We can further combine the two approaches in a straightforward manner. Once we decided on a space parameter b_1 we can first find a tree-decomposition with separators bounded by b as suggested by the super-clustering approach. The decomposition suggests a collection of problems that can be solved by search. Instead, we can reduce the time complexity further by applying elim-cond(b_2) in each subproblem, independently. Clearly we must

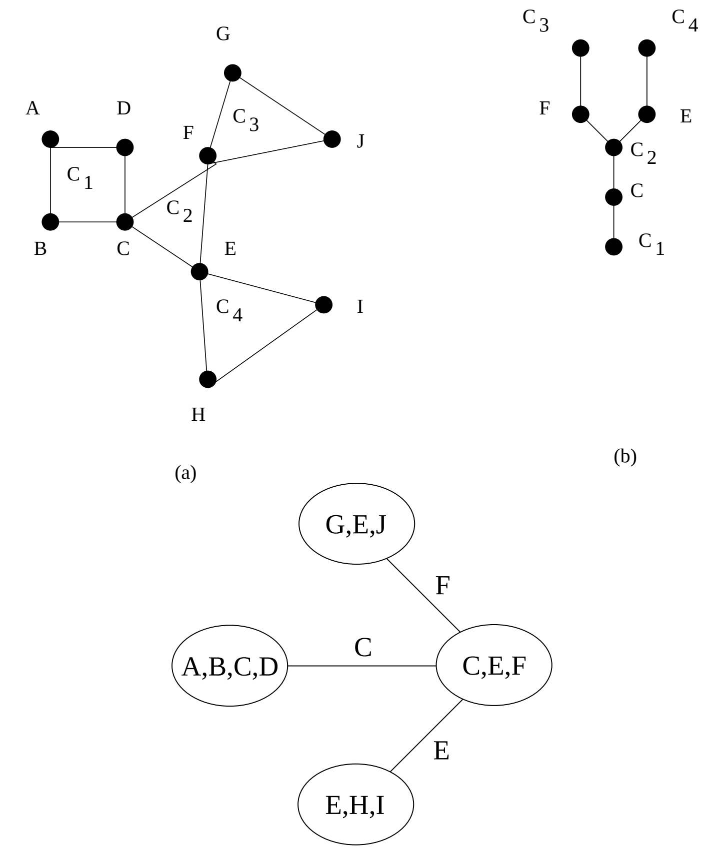


Figure 10.14: A graph and its decomposition into non-separable components.

have $b_2 \leq b_1$ to maintain the space requirement, but the time complexity in each clique can be reduced significantly as a result.

We call the combined algorithm HYBRID(b_1, b_2). First a tree-decomposition having separators bounded by b_1 are created and subsequently the CTE algorithm is applied, but each clique is processed by elim-cond(b_2). If c_b^* is the size of the maximum b_2-cutset in each clique of the b-tree-decomposition, the resulting algorithm is space exponential in b but time exponential in c_b^*.

Two special cases are worth mentioning. The first is when we apply the cycle-cutset scheme in each clique. This method was tested empirically on structured networks that are associated with real circuits (for circuit diagnosis tasks) and the reduction in complexity bounds for complex circuits was shown to be tremendous [44].

The second special case is when $b_1 = b_2$. For $b = 1$ hybrid(1,1) corresponds to applying the non-separable components as a tree-decomposition and utilizing the cycle-cutset in each component. The space complexity of this algorithm is linear but its time com-

$bucket_{(GJF)}$: $R_{GF}, R_{GJ}, R_{JF}, D_G, D_F$
$bucket_{(EHI)}$: $R_{EHI}, D_I,$
$bucket_{(CFE)}$: R_{CFE}
$bucket_{(ABCD)}$: $R_{AD}, R_{AB}, R_{DC}, R_{BC}$

$bucket_{(GJF)}$: $R_{GF}, R_{GJ}, R_{JF}, D_G, D_F \mid\mid \rho^{GJF}_{CFE}(F)$
$bucket_{(EHI)}$: $R_{EHI}, D_I, \mid\mid \rho^{EHI}_{CFE}(E)$
$bucket_{(CFE)}$: $R_{CFE} \mid\mid , \rho^{CFE}_{GJF}(F), \rho^{CFE}_{EHI}(E) \mid\mid \rho^{CFE}_{ABCD}(C)$
$bucket_{(ABCD)}$: $R_{AD}, R_{AB}, R_{DC}, R_{BC} \mid\mid , \rho^{ABCD}_{CFE}(C)$

Figure 10.15: A schematic execution of ATC when the superbucket are the non-separable components. (a) the original superbucket tree decomposition (b) the super-bucket tree including the messages received. R denotes original constraints and ρ denotes new domain constraints

plexity can be much better than the cycle-cutset scheme or the non-separable component approach alone.

10.4 A case study of combinatorial circuits

It was shown that a variety of diagnosis and testing tasks over combinatorial circuits can be modeled as constaint processing or as constraint optimization tasks. Therefore, such circuits can shed light on the structural parameters associated with one real benchmark. The anlysis uses benchmark combinatorial circuits, widely used in the fault diagnosis and testing community [18] (See Table 10.1.) . None of the circuits are trees and they all have considerable fanout nodes as shown in the schematic diagram of circuit c432 in Fig. 10.16.

Each circuits is associated with a primal graph having a node for each variable in the circuit. For every gate in the circuit, the graph has edges connecting all the variables of the gate's inputs and output. Table 10.2 gives the number of nodes and edges of the primal graph for each circuit.

Tree decompositions of each circuit graph was generated using the triangulation approach: by selecting an ordering for the nodes, then triangulating the graph and identifying its maximum cliques. Several ordering heuristics were considered including maximum-cardinality, minimum-width, and minimum-degree and the results are reprted using the min-degree ordering which was by far the best, yielding the smallest cliques sizes and separators. For results on the other orderings see [44].

Table 10.1: ISCAS '85 Benchmark Circuit Characteristics

Circuit Name	Circuit Function	Total Gates	Input Lines	Output Lines
C17		6	5	2
C432	Priority Decoder	160 (18 EXOR)	36	7
C499	ECAT	202 (104 EXOR)	41	32
C880	ALU and Control	383	60	26
C1355	ECAT	546	41	32
C1908	ECAT	880	33	25
C2670	ALU and Control	1193	233	140
C3540	ALU and Control	1669	50	22
C5315	ALU and Selector	2307	178	123
C6288	16-bit Multiplier	2406	32	32
C7552	ALU and Control	3512	207	108

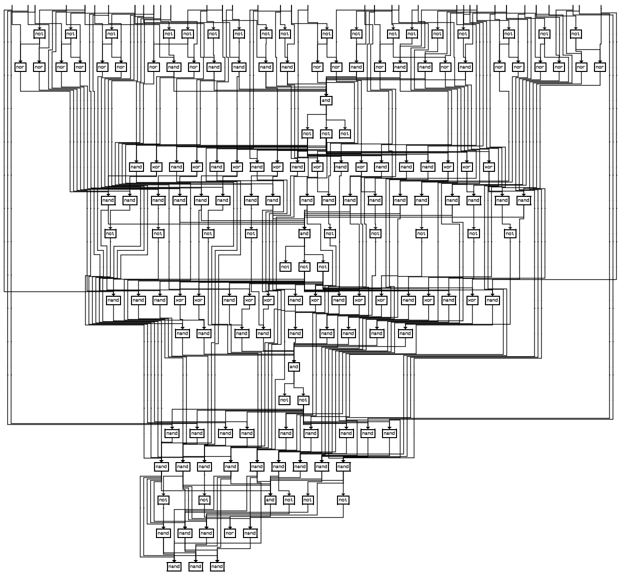


Figure 10.16: Schematic of circuit c432: 36 inputs 7 outputs and 160 components.

Table 10.2: Number of nodes and edges for the primal graphs of the circuits.

Circuit	c17	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552
#nodes	11	196	243	443	587	913	1426	1719	2485	2448	3719
#edges	18	660	692	1140	1660	2507	3226	4787	7320	7184	9572

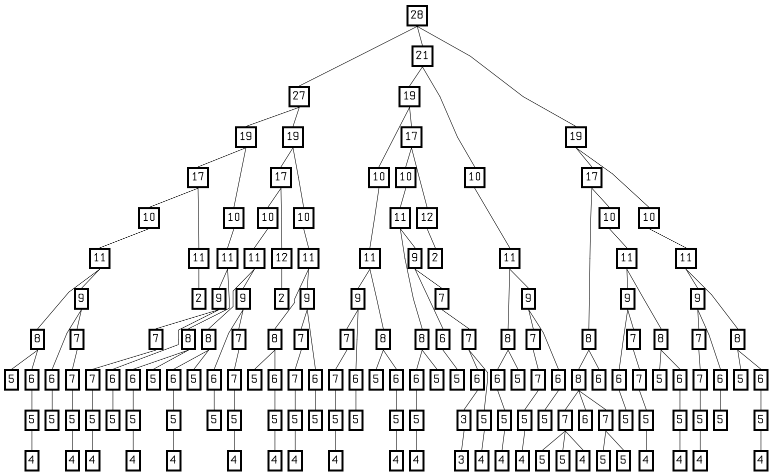


Figure 10.17: Primary join tree (157 cliques) for circuit c432 (196 variables); the maximum separator size is 23.

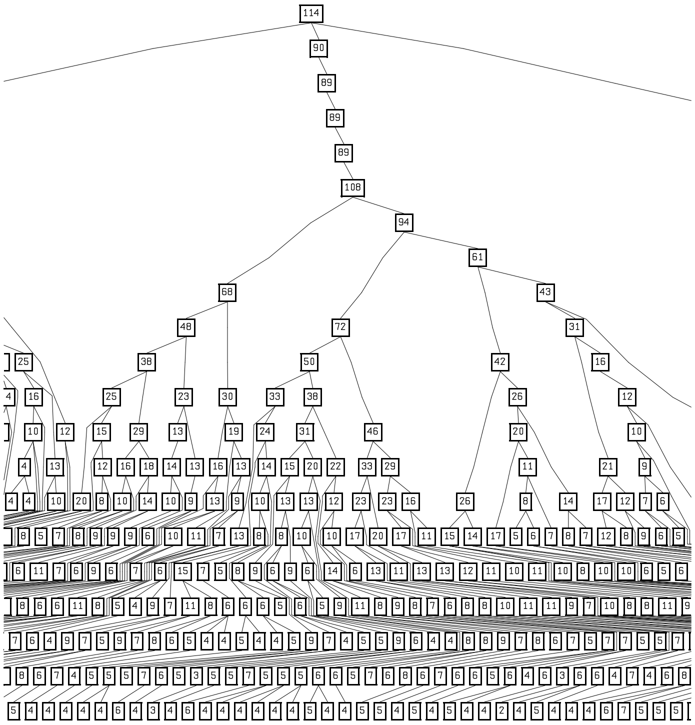


Figure 10.18: Part of primary join tree (1419 cliques) for circuit c3540 (1719 variables) showing the descendants of the root node down to the leaves; the maximum separator size is 89.

10.4.1 Parameters of primary join trees

For each primary join tree generated, three parameters are computed: (1) the size of cliques, (2) the size of cycle-cutsets in each of the subgraphs defined by the cliques, and (3) the size of the separator sets. The nodes of the join tree are labeled by the cliques (or clusters) sizes and the edges are labeled by the separator size. We present the results on circuit c432 only having 196 variables. Results on other circuits are summarized in [44].

Figure 10.17 and 10.18 present information on the primary join tree, For example, Figure 10.17 shows that the cliques sizes range from 2 to 28. The root node has 28 nodes and the descendant nodes have strictly smaller sizes. The depth of the tree is 11 and all nodes whose distance from the root is greater than 6 have sizes strictly less than 10. The leaves have sizes ranging from 2 to 6. The corresponding numbers for the primary join tree of the larger circuit c3540 shown in Fig. 10.18.

The structural analysis can suggest the parameters associated with either complete inference or with the hybrid elim-cond(1) on that constraint problems which is equal to the cycle-custet scheme.

As we see, some cliques and separators require memory space exponential in 23 for circuit c432 and exponential in 89 for circuit c3540. This is clearly not feasible. We will next evaluate the potential of $hybrid(b, 1)$.

10.4.2 Parameters controling hybrids

Let s_0, c_0 be the maximum cutset and separator size of the primary join tree T_0 obtained by tree clustering. Let $s_0, s_1, \ldots, s_n$ be the size of the separators in T_0 listed from largest to smallest. As explained earlier, with each separator size, s_i, we associate a tree decomposition T_i generated by combining adjacent clusters whose separators' sizes are strictly larger than s_i. We denote by c_i the largest cutset size in any cluster of T_i.

Figures 10.19 provides information regarding the tree-width, and cycle-custest size as a function of the different separators in the ceconday tree-decompositions.

Figure 10.20 gives a chart presenting bounds for time versus space for each circuit. Each point in the chart corresponds to a specific secondary join tree decomposition T_i and has the space complexity measured by the separator size, s_i, and the time complexity by the cutset size in each clique.

Each chart in Figure 10.20 can be used to select the parameter b that controls best the balance between time and space. We see the gradual effect of lowering space on the time required by a corresponding clustering+conditioning algorithm. For example circuit c432 (Figure 10.20) shows the separator size (space) which is initially 23 (for the primary join tree) gradually reduced down to 1 in a series of secondary trees. The figure demonstrates that reducing the separator size (to meet the space restrictions) increases the worst-case

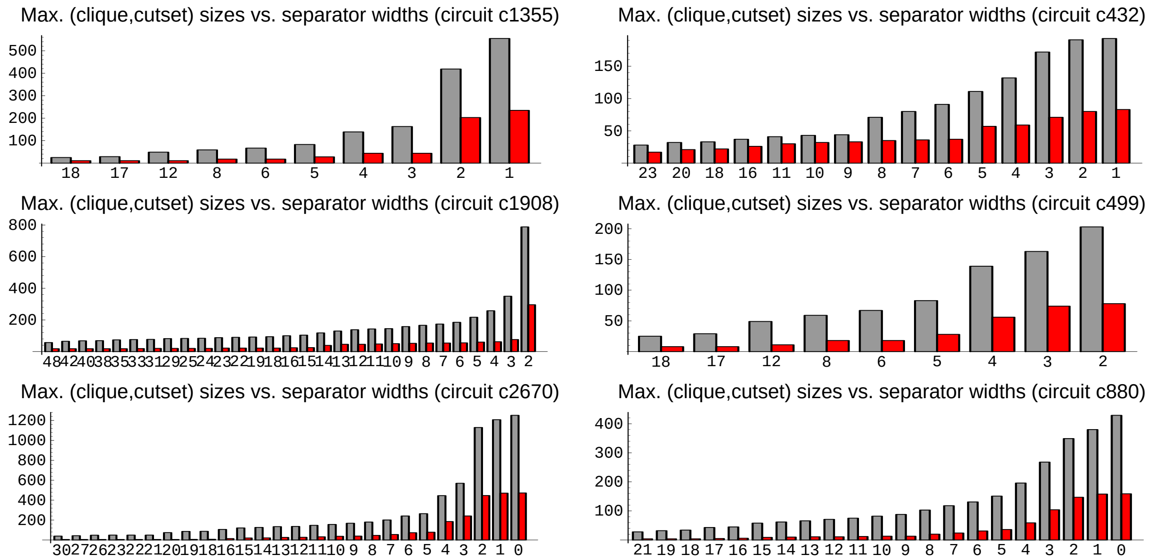


Figure 10.19: Size of max clicues and max cutsets vs separator sizes in secondary trees

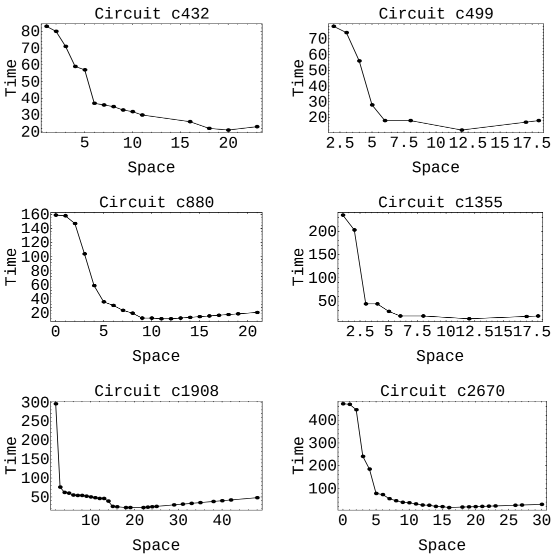


Figure 10.20: Time/Space tradeoff for c432 (196 variables), c499 (243 variables), c880 (443 variables), c1355 (587 variables), c1908 (913 variables) and c2670 (1426 variables). Time is measured by the maximum of the separator size and the cutset size and space by the maximum separator size.

time complexity of the hybrid algorithm. The time increases because of the large clusters contained in the secondary join tree and the corresponding increase in the size of cutsets.

Figure 10.21 display the structure of secondary join trees for c432. The primary join tree for the circuit is shown in Figure 10.17. The secondary trees are indexed by the separator sizes of the primary tree which range from 1 to 23 (Figure 10.21). As the separator decreases the maximum clique size increases, and both the size and the depth of the tree decrease. Like the primary join tree, each secondary join tree has also a skewed distribution of the clique sizes. Note that the clique size for the root node is significantly larger than for all other nodes, and is increasing as the separator decreases.

10.5 Summary

Problem solving methods can be viewed as hybrids of two main principles: inference and search. Tree-decomposition and variable-elimination is an example of an inference algorithm while backtracking is an example of a search method. Variable-elimination algorithms are time and space exponential in the size of the induced width while search algorithms are time exponential but require only linear memory. The chapter presents a structure-based hybrid scheme that uses inference and search as its two extremes and, using a single design parameter, permits the user to control and tailor the storage-time tradeoff in accordance with the problem domain and the available resources

The structural parameters of interest are: (1) the size of cliques in a tree-decomposition, namely, the induced-width, (2) the size of cycle-cutsets in each of the subgraphs defined by the cliques, and (3) the size of the separator sets.

The hybrid is allowed by observing the conditioning and elimination different effect on the constraint graph. Assigning x_i a value (conditioning on x_i) corresponds to deleting it from the constraint graph, while eliminating x_i require connecting all its parents and then eliminating the corresponding node from the graph. Indeed, in bucket-elimination, when x_i is assigned a value, no new functional dependencies are created, while processing the bucket of x_i adds new components to the buckets of x_i's parents. We focus on such framework for both constraint satisfaction and propositional satisfiability.

Two architectures for hybrids are proposed. The *conditioning-first* idea is to find a small subset of *conditioning-cutset* variables Y, such that conditioning (that is, assigning them values) result in a subproblem having low w^* graph. The algorithm ELIM-COND(B) searches the space of partial assignments over the b-cutset variables and solves by variable elimination the rest of the problem. The special case of conditioning on a cycle-cutset is the cycle-cutset decomposition. The second, inference-first, approach is to limit space by using tree-decompositions whose separators are bounded by b. In particular the tree of nonseparable components is a special case whose separators' size equals 1, yielding a

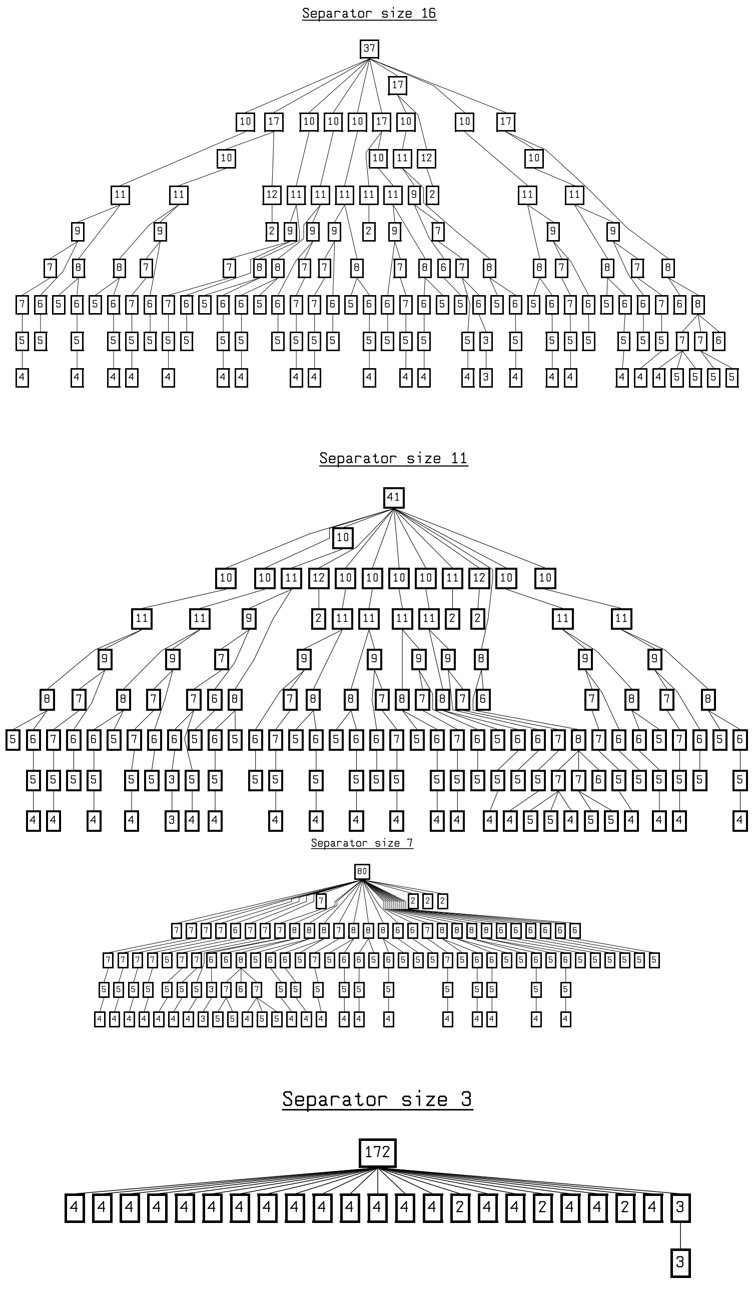


Figure 10.21: Secondary trees for c432 with separator sizes 16 and 11 7 and 3.

linear space algorithm. Both hybrid algorithms allow controlled tradeoff between time and space.

Finally, a hybrid of the two hybrids allows even better time complexity for every fixed space restrictions.

The chapter also demonstrates the applicability of the scheme to real-life domains through the structural parameters of benchmark circuits widely used in the fault diagnosis and testing community [18].

10.6 Chapter notes

The cycle-cutset scheme for probabilistic inference was introduced by Pearl [141] and for constraint networks by Dechter and Pearl [46, 39].

This idea of extending the combination of conditioning and tree-clustering beyond the cycle-cutset scheme appears in the work of Jegou [90] for constraint networks and in the works of [146, 78] for probabilistic networks. In the work of Jegou various heuristic are presented, aiming at creating a hybrid algorithm having improved time performance. In [78] the idea is applied to the probabilistic infernce system pathfinder system, when the conditioning set is restricted to the set of diseases.

Hybrid algorithms of inference and search were first introduced using the cycle-cutset decomposition [46, 39] for constraints and also were termed loop-cutset in the area of probabilistic inference [141]. Extension of this approach to the general case were presented in propositional cnfs [148], and for optimization tasks [109], as well as by [33]. The inference-first idea for trading space for time was introduced in [44].

These hybrids suggested a secondary optimization tasks of finding the minimal-cutset in graphs. The problem is also known as minimal feedback set and was shown to be NP-complete. Approximation algorithms were investigated by Becker et. al. [2]. It would be interesting to extend such algorithms for the more general case of finding the minimal b-cutset of a graph.

10.7 Exercises

1. (research question) Develop approximation algorithms for finding a minimal b-cutset for an given constant b.

2. Prove that deciding the consistency of a constraint network whose constraint graph has non-separable components of size at most r can be solved in time exponential in r while in linear space.

3. Consider the crossword puzzle described in Chapter 2, formulated in the dual representation.

   (a) Bound the complexity of solving the problem using the cycle-cutset algorithm. Show a trace of the cycle-cutset algorithm of up to a page in length.

   (b) Bound the complexity of solving the problem using the non-separable component algorithm. Show a trace of CTE applied to such a tree-decomposition of up to a page in length.

   (c) Compare the above approaches (tree-clustering, bi-component, and cycle-cutset) for solving the problem in terms of time and space complexity.