Topological Parameters for Time-Space Tradeoff

Rina Dechter
Information & Computer Science
University of California
Irvine, CA 92717
dechter@ics.uci.edu

Yousri El Fattah
Rockwell Science Center
1049 Camino Dos Rios
Thousand Oaks, CA 91360
yousri@rsc.rockwell.com

March 17, 2000

Abstract

In this paper we propose a family of algorithms combining tree-clustering with conditioning that trade space for time. Such algorithms are useful for reasoning in probabilistic and deterministic networks as well as for accomplishing optimization tasks. By analyzing the problem structure the user can select from a spectrum of algorithms, the one that best meets a given time-space specification. To determine the potential of this approach, we analyze the structural properties of problems coming from the circuit diagnosis domain. The analysis demonstrate how the tradeoffs associated with various hybrids can be explicated and be used for each problem instance.

1 Introduction

Problem solving methods can be viewed as hybrids of two main principles: inference and search. Tree-clustering is an example to an inference algorithm while the cycle-cutset is an example of a search method. Tree clustering algorithms are time and space exponential in the size of their cliques while search algorithms are time exponential but require only linear memory. In this paper we developed a hybrid scheme that uses inference (tree-clustering) and search (cycle-cutset conditioning) as its two extremes and, using a single, structure-based design parameter, permits the user to control the storage-time tradeoff in accordance with its problem domain and the available resources.

Topology-based algorithms for constraint satisfaction and probabilistic reasoning fall into two distinct classes. One class is centered on tree-clustering, the other on cycle-cutset decomposition. Tree-clustering involves transforming the original problem into a tree-like problem [26, 22, 14] that can then be solved by a specialized efficient tree-solving algorithm [27, 28]. The transforming algorithm identifies subproblems that together form a tree, and the solutions to the subproblems serve as the new values of variables in a tree metalevel problem. The metalevel problem is called a *join-tree*. The tree-clustering algorithm is time and space exponential in the tree-width of the problem's graph. A related parameter is the *induced-width* which equals the tree-width. We will use both terms interchangeably.

The *cycle-cutset* method, also called *loop-cutset conditioning*, utilizes the problem's structure in a different way. It exploit the fact that variable instantiation changes the effective connectivity of the underlying graph. A cycle-cutset of an undirected graph is a subset of its nodes which, once removed, cuts all of the graph's cycles. A typical cycle-cutset method enumerates the possible assignments to a set of cutset variables and, for each cutset assignment, solves (or reasons about) a tree-like problem in polynomial time. Thus, the overall time complexity is exponential in the size of the cycle-cutset [8, 29, 15]. Fortunately, enumerating all the cutset's assignments can be accomplished in linear space, yielding an overall linear space algorithm.

The first question is which method, tree-clustering or the cycle-cutset scheme provide a better worst-case time guarantees. This question was answered by Bertele and Briochi [5] in 1972 and later reaffirmed in [31]. They showed that, the minimal cycle-cutset of any graph can be much larger, and is never smaller than its minimal tree-width. In fact, for an arbitrary graph, $r \leq c + 1$, where c is the minimal cycle-cutset and r is the tree-width [5]. Consequently, for any problem instance the time guarantees accompanying the cycle-cutset scheme are never tighter than those of tree-clustering, and can be even much worse. On the other hand, while tree-clustering requires exponential space (in the induced-width) the cycle-cutset requires only linear space.

Since the space complexity of tree-clustering can severely limit its usefulness, we investigate in this paper the extent to which space complexity can be reduced, while reasonable time complexity guarantees are maintained. Is it possible to have the time guarantees of clustering while using linear space? On some problem instances, it is possible. Specifically, on those problems whose associated graph has an induced width and a cycle-cutset of comparable sizes (e.g., on a ring, the cutset size is 1 and the tree width is 2, leading to identical time bounds). We conjecture, however, that any algorithm that has a time bound guarantee exponential in the induced-width will, on some problem instances, require exponential space in the induced width.

The space complexity of tree-clustering can be bounded more tightly using the *separator size*, which is defined as the size of the maximum subset of variables shared by adjacent subproblems in the join-tree. Indeed some of the variants of tree-clustering algorithms send messages over the separators only and therefore can comply with space complexity that is exponential in the separator size only [22, 38]. Our investigation employs the separator size to control the time-space tradeoff. The idea is to combine adjacent subproblems joined by a large separator into one bigger cluster or subproblem so that the remaining separators are of smaller size. Once a join-tree with smaller separators is generated, its potentially larger clusters can be solved using the cycle-cutset method, or any other linear-space scheme.

In this paper we will develop a time-space tradeoff scheme that is applicable to belief network processing, constraint processing, and optimization tasks, yielding a sequence of parameterized algorithms that can trade space for time. With this scheme it will be possible to select from a spectrum of algorithms the one that best meets some time-space requirement. Algorithm tree-clustering and cycle-cutset conditioning are two extremes in this spectrum.

We investigate the potential of our scheme in the domain of combinatorial circuits. This domain is frequently used as an application area in both probabilistic and deterministic reasoning [18, 40, 16]. We analyze 11 benchmark combinatorial circuits widely used in the fault diagnosis and testing community [6] (see Table 1 ahead.). For each circuit, the analysis is summarized in a chart displaying the time-space complexity tradeoffs for diagnosing that circuit. The analysis allows tailoring the hybrid of tree-clustering and cycle-cutset decomposition to the available memory.

Different variants of the tree-clustering transformation algorithms were developed in recent years, both for constraint processing ([11]) and for probabilistic inference [29, 26, 22, 37, 38, 24, 23]. Some of these algorithms compile functions over the cliques and some restrict this information to the separators. Some are directional, query-based while others are symmetrical, compiling answers for a variety of queries. We choose to demonstrate our idea using a directional variant of tree-clustering that is query-based, which we hope simplify the exposition. The approach is applicable to any variant of tree-clustering.

Section 2 gives definitions and preliminaries and introduces the time-space tradeoff ideas for belief networks. Sections 3 and 4 briefly extend these ideas to constraint networks and to optimization problems. Section 5 and 6 describe the empirical analysis and the results. Section 7 discusses related work and Section 8 gives our conclusions.

The paper assumes familiarity with the basic concepts of tree-clustering and cycle-cutset conditioning and provides only brief necessary background. For more details the reader should consult the references.

2 Probabilistic Networks

2.1 Overview

2.1.1 Definitions and notations

Belief networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. It is defined by a directed acyclic graph over nodes representing random variables of interest (e.g., the temperature of a device, the gender of a patient, a feature of an object, the occurrence of an event). The arcs signify the existence of direct causal influences between the linked variables. The strength of these influences are quantified by conditional probabilities that are attached to each cluster of parents-child nodes in the network. A belief network is a concise description of a complete probability distribution. It uses the concept of a directed graph.

Definition 1 [Directed graph] A *directed graph* $G = \{V, E\}$, where $V = \{X_1, ..., X_n\}$ is a set of elements and $E = \{(X_i, X_j)|X_i, X_j \in V\}$ is the set of edges. If an arc $(X_i, X_j) \in E$, we say that X_i points to X_j. For each variable X_i, $pa(X_i)$ is the set of variables pointing to X_i in G, while $ch(X_i)$ is the set of variables that X_i points to. The family of X_i includes X_i and its parent variables. A directed graph is acyclic if it has no directed cycles. In an undirected graph the direction of the arcs is ignored: (X_i, X_j) and (X_j, X_i) are identical. An undirected graph is chordal if every cycle of length at least 4 has a chord. A clique is a subgraph that is completely connected and a maximal clique of graph is a clique that is not contained in any other clique of the graph.

Definition 2 [Belief Networks] Let $X = \{X_1, ..., X_n\}$ be a set of random variables over multi-valued domains, $D_1, ..., D_n$. A *belief network* is a pair (G, P) where G is a directed acyclic graph over the nodes X and $P = \{P_i\}$ are the conditional probability matrices over the families of G, $P_i = P(X_i|pa(X_i))$. An assignment $(X_1 = x_1, ..., X_n = x_n)$ can be abbreviated as $x = (x_1, ..., x_n)$. The belief network represents a probability distribution over X having the product form

$$P(x_1, ..., x_n) = \Pi_{i=1}^n P(x_i|x_{pa(X_i)})$$

where $x_{pa(X_i)}$ denotes the projection of a tuple x over $pa(X_i)$. An evidence set e is an instantiated subset of variables. A *moral graph* of a belief network is an undirected graph generated by connecting the tails of any two head-to-head pointing arcs in G and removing the arrows. A belief network is a polytree if its underlying undirected (unmoralized) graph has no cycles (namely, it is a tree).

Definition 3 [Induced width, induced graph] An *ordered graph* is a pair (G, d) where G is an undirected graph and $d = X_1, ..., X_n$ is an ordering of the nodes. The *width of a node* in an ordered graph is the number of its earlier neighbors. The *width w(d) of an ordering d*, is the maximum width over all nodes. The *induced width of an ordered graph*, $w^*(d)$, is the width of the induced ordered graph obtained by processing the nodes recursively, from last to first; when node X is processed, all its earlier neighbors are connected. This process is also called "triangulation". The induced (triangulated) graph is clearly chordal. The *induced width of a graph*, $w*$, is the minimal induced width over all its orderings (For more information see [13, 9].

Example 1: Figure 1 shows a belief network's acyclic graph and its associated moral graph. The width of the graph in Figure 1b along the ordering $d = A, B, C, D, G, E, F, H$ is 3. Since the moral graph in Figure 1(b) is chordal no arc is added when generating the induced ordered graph. Therefore, the induced-width w^* of the graph is also 3.

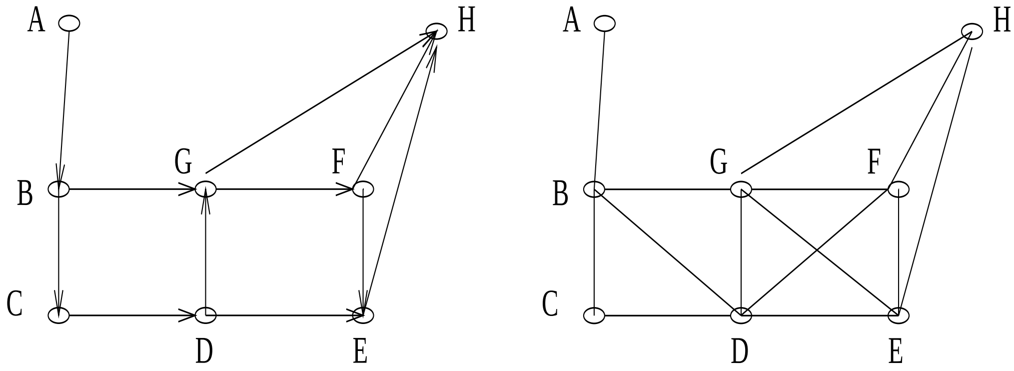

Figure 1: (a) A belief network and (b) its moral graph

The most common task over belief networks is to determine posterior beliefs of some variables. Other important tasks are *mpe*: finding the most probable explanation given a set of observations, or *map*: finding the maximum aposteriori hypotheses given evidence, both tasks are relevant to abduction and diagnosis [29]. It is well known that such tasks can be answered effectively for singly-connected polytrees by a belief propagation algorithm [29] that can be extended to multiply-connected networks by either tree-clustering or loop-cutset conditioning.

2.1.2 Tree-Clustering

The most widely used method for processing belief networks is join-tree clustering. The algorithm transforms the original network into a tree of subproblems called *join-tree*. Tree-clustering methods have two parts. In the first part the structure of the newly generated tree problem is decided, and in the second part the conditional probabilities between the subproblems, (viewed as high-dimensional variables) is determined. The structure of the join-tree is determined primarily by graph information, embedding the graph in a tree of cliques as follows. First the moral graph is embedded in a chordal graph by adding some edges. This is accomplished by picking a variable ordering $d = X_1, ..., X_n$, then, moving from X_n to X_1, recursively, connecting all the earlier neighbors of X_i in the moral graph yielding the induced ordered graph. Its *induced width* $w^*(d)$, as defined earlier, is the maximal number of earlier neighbors each node has.

Clearly, each node and its earlier neighbors in the induced graph is a clique. The maximal cliques, indexed by their latest variable in the ordering, can be connected into a *clique-tree* and serve as the subproblems (or clusters) in the final join-tree. The clique-tree is created by connecting every clique C_i to an earlier clique C_j, called its parent, with whom it shares a maximal number of variables. Clearly, the induced width $w^*(d)$ equals the size of the maximal clique minus 1.

Once the join-tree structure is determined, each conditional probability table (CPT) is placed in a clique containing all its arguments. The marginal probability distributions for each clique can then be computed by multiplying all the CPTs and normalizing, and subsequently the conditional probabilities between every clique and its parent clique can be derived. Tree-clustering, therefore, is time and space exponential in the size of the maximal clique, namely, exponential in the moral graph's induced-width (plus 1). In Figure 1b, the maximal cliques of the chordal graph are $\{(A, B), (B, C, D), (B, D, G), (G, D, E, F), (H, G, F, E)\}$, resulting in the join-tree structure given in Figure 3(a).

Clearly, the established connection between induced-width and complexity, motivates finding an ordering with a smallest induced-width, a task known to be hard [2, 1]. However, useful greedy heuristics as well as approximation algorithms were developed in the last decade, taking into account not only graph information, but also the variance in the variables domain sizes [14, 5, 29, 26, 4, 39, 24].

A tighter bound on the space complexity of tree-clustering may be obtained using the *separator size*. The separator size of a join-tree is the maximal size of the intersections between any two cliques, and the *separator size of a graph* is the minimal separator size over all the graph's join-trees [7].

Algorithm *Directional tree clustering (DTC)*, presented in Figure 2 is a query-based variant of join-tree clustering. It records functions on separators only. The algorithm can be viewed as rephrasing one phase (the collect phase) in both the Shafer-Shenoy architecture [38, 36], as well as the one phase propagation in the Hugin architecture. The Hugin architecture has its roots in the method proposed by Lauritzen and Spiegelhalter [26] for computing marginals of probability distributions. It was proposed by Jensen et. al. [22] and is incorporated in the software product Hugin.

Step 1 of algorithm DTC computes the join-tree by first determining the cliques and connecting them in a tree-structure where each cliques has a parent clique and a parent separator. Once the structuring part of the join-tree is determined, each CPT is placed in one clique that contains its arguments. For example, given the join-tree structure in Figure 3a, the cliques contain the CPTs as follows. Clique AB contains $P(B|A)$; BCD contains $P(C|B)$ and $P(D|C)$; BDG contains $P(G|B,D)$; $GDEF$ contains $P(E|D,F)$ and $P(F|G)$; and finally, clique $GEFH$ contains $P(H|G,F,E)$. Subsequently, algorithm DTC processes the cliques recursively from leaves to the root. Processing a clique involves computing the product of all the probabilistic functions that reside in that clique and then summing over all the variables that do not appear in its parent separator. The computed function (over the parent separator) is added to the parent clique. Computation terminates at the root-clique[1].

As shown ahead, the algorithm tightens the bound on space complexity using its separator size. This modification to its space management can be applied to any variant of tree-clustering which is not necessarily query-based. In summary,

Theorem 1: [time-space of join-tree clustering] Given a belief network having n variables, whose moral graph can be embedded in a clique-tree having induced width r and separator size s, the time complexity for determining beliefs and the mpe by a join-tree clustering algorithm (e.g., by DTC) is $O(n \cdot exp(r))$ while its space complexity is $O(n \cdot exp(s))$.

Proof: It is well known that the time and space complexity of join-tree clustering is bounded by the induced-width (clique sizes) of the graph, r. The only thing that need to be shown, therefore, is that the tighter bound on space complexity is valid. Consider the processing of a clique (step 4 in DTC). Let C be the variables in the clique, let S be the variables in the separator with its parent clique and let $U = C - S$. In step 4 we compute $\lambda = \sum_U \Pi_{i=1}^{j} \lambda_i$. Namely, λ_p is a function defined over S, because all the variables in U are eliminated by summation. Namely, for each assignment s to S, $\lambda(s)$ can be computed in linear space as follows: we initialize $\lambda(s) \leftarrow 0$, and then for every assignment u to U we compute the running sum: $\lambda(s) \leftarrow \lambda(s) + \Pi_i \lambda_i(s,u)$. $\square$.

[1] We disregard algorithmic details that do not affect asymptotic worst-case analysis here.

Algorithm directional join-tree clustering (DTC)
Input: A belief network (G, P), where G is a DAG and $P = \{P_1, ..., P_n\}$,
Output: the belief of X_1 given evidence e.

1. Generate a join-tree clustering of G, identified by its cliques $C_1, ...C_t$. Place each P_i and each observation in one clique that contains its arguments (One's favorite structuring into tree-clustering methods can be used.)

2. Impose directionality on the join-tree, namely create a rooted directed tree whose root is a clique containing the queried variable. Let $d = C_1, ...C_t$ be a breadth-first ordering of the rooted clique-tree, let $S_{p(i)}$ and $C_{p(i)}$ be the parent separator and the parent clique of C_i, respectively.

3. From $i \leftarrow t$ downto 1 do

4. (Processing clique C_i):
   Let $\lambda_1, \lambda_2, ..., \lambda_j$ be the functions in clique C_i, and when C_i denotes also its set of variables,

   - For any observation $X_j = x_j$ in clique C_i substitute x_j in each function in the clique.
   - Let $U_i = C_i - S_{p(i)}$ and u_i is an assignment to U_i. compute
     $\lambda_p = \sum_{u_i} \Pi_{i=1}^{j} \lambda_i$.
     Put λ_p in parent clique $C_{p(i)}$.

5. **Return** (processing root-clique, C_1), $Bel(x_1) = \alpha \sum_{u_1} \Pi_i \lambda_i$
   α is a normalizing constant and u_1 is an assignment to $U_1 = C_1$.

Figure 2: Algorithm directional join-tree clustering

Clearly $s \leq r$. Note that since in the example of Figure 1 the separator size is 3 and the induced width is also 3, we do not gain much space-wise, by the modified algorithm. There are, however, many cases where the separator size is much smaller than the induced-width.

Algorithm directional tree-clustering (DTC), can be adapted for the task of finding the most probable explanation (mpe), by replacing the summation operation by maximization.

2.1.3 Cycle-cutset conditioning

Belief networks may be processed also by cutset conditioning [29]. A subset of nodes of an undirected graph is called a *cycle-cutset*, if removing all the edges incident to nodes in the cutset makes the graph cycle-free. A subset of nodes of an acyclic-directed graph is called a *loop-cutset* if removing all the outgoing edges of nodes in the cutset results in a poly-tree [29, 30]. A minimal cycle-cutset (resp. minimal loop-cutset) is such that if one node is removed from the set, the set is no longer a cycle-cutset (resp., a loop-cutset).

Algorithm cycle-cutset -conditioning (also called cycle-cutset decomposition or loop-cutset conditioning) is based on the observation that assigning a value to a variable changes the effective C:W connectivity of the network. Graphically this amounts to removing all outgoing arcs from the assigned variables. Consequently, an assignment to a subset of variables that constitute a loop-cutset means that belief updating, conditioned on this assignment, can be carried out in the resulting poly-tree [29]. Multiply-connected belief networks can therefore be processed by enumerating all possible instantiations of a loop-cutset and solving each conditioned network using the poly-tree algorithm. Subsequently, the conditioned beliefs are combined using a weighted sum where the weights are the probabilities of the joint assignments to the loop-cutset variables conditioned on the evidence. Pearl [29] showed that weights computation is not more costly than enumerating all the conditioned beliefs.

This scheme was later simplified by Peot and Shachter [30]. They showed that if the polytree algorithm is modified to compute the probability of each variable-value proposition *conjoined* with the evidence, rather than *conditioned* on the evidence, the weighted sum can be replaced by a simple sum.

In other words:

$$P(x|e) = \alpha P(x,e) = \alpha \sum_c P(x,e,c)$$

If $\{X\} \cup C \cup E$ is a loop-cutset (note that C and E denote subsets of variables) then $P(x,e,c)$ can be computed very efficiently using a propagation-like algorithm on poly-trees. Consequently the complexity of the cycle-cutset scheme is exponential in the size of C where $C \cup \{X\} \cup E$ is a loop-cutset. Some additional improvements are presented in [15]. In summary,

Theorem 2: ([29, 30]) Given a belief network (G, P) having n variables, and family sizes bounded by $|F|$, and a loop-cutset bounded by c, belief updating and mpe can be computed in time $O(n|F| \cdot exp(c))$ and in linear space.[2] □

2.2 Trading Space for Time

Assume now that we have a problem whose join-tree has induced width r and separator size s but space restrictions do not allow the necessary $O(exp(s))$ memory required by tree-clustering. One way to overcome this problem is to collapse cliques joined by large separators into one big cluster. The resulting join-tree has larger subproblems but smaller separators. This yields a sequence of tree-decomposition algorithms parameterized by the sizes of their separators.

Definition 4 [Primary and secondary join-trees]: Let T be a clique-tree embedding of the moral graph of G. 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 clusters whose separator sizes are strictly greater than s_i. $T = T_0$ is called the primary join-tree, while T_i, when $i > 0$, is a secondary join-tree. We denote by r_i the largest cluster size in T_i.

Note that as s_i decreases, r_i increases. Clearly, from Theorem 1 it follows that

[2] Another bound often used is $O(n \cdot exp(c+2))$ where the "2" in the exponent comes from the fact that belief updating on binary trees is linear in the size of the CPTs between pairs of variables which are at least $O(k^2)$.

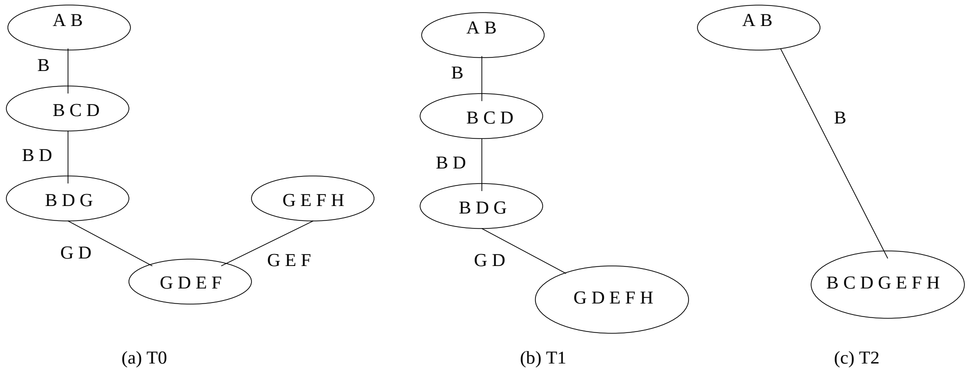

Figure 3: A tree-decomposition with separators equal to (a) 3, (b) 2, and (c) 1

Theorem 3: Given a join-tree T over n variables, having separator sizes $s_0, s_1, ..., s_t$ and corresponding secondary join-trees having maximal clusters, $r_0, r_1, ..., r_t$, belief updating and mpe can be computed using any one of the following time and space bounds $O(n \cdot exp(r_i))$ time, and $O(n \cdot exp(s_i))$ space, (i range over all the of secondary join-trees), respectively.

Proof: For each i, a secondary tree T_i is a structure underlying a possible execution of directional join-tree clustering. From Theorem 1 it follows that the time complexity is bounded exponentially by the corresponding cliques size (e.g., r_i) and space complexity is bounded exponentially by the corresponding separator, s_i. $\square$

Example 2: If in our example, we allow only separators of size 2, we get the join tree T_1 in Figure 3(b). This structure suggests that we can update beliefs and compute mpe 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 3(c). We conclude that the problem can be solved, either in $O(k^4)$ time (k being the maximum domain sizes) and $O(k^3)$ space using the primary tree 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.

We know that finding the smallest induced width of a graph (or finding a join-tree having smallest cliques) is NP-complete [2, 35]. Nevertheless, many greedy ordering algorithms provide useful upper bounds. We denote by w_s^* the smallest induced width among all the tree embeddings of G whose separators are of size s or less. Finding w_s^* may be hard as well, however. We can conclude that: Given a belief network BN, for any $s \leq n$, if $O(exp(s))$ space can be used, then belief updating and mpe can, potentially be computed in time $O(exp(w_s^* + 1))$.

2.2.1 Using the cycle-cutset scheme within cliques

Finally, instead of executing a brute-force algorithm to compute the marginal distributions over the separators (see step 4 in DTC), we can use the loop-cutset scheme. Given a clique C_p with a separator parent set S_p, step 4 computes a function defined over the separator, by

$$\lambda_p = \sum_{u_p} \prod_{i=1}^{j} \lambda_i$$

where $U_p = C_p - S_p$. This seems to suggest that we have to enumerate explicitly all tuples over C_p. However, we observe that when computing λ_p for a particular value assignment of the separator x_s, those assignments can be viewed as cycle-breaking values in the graph. So, when the separator constitutes a loop-cutset then the sum can be computed in linear time, either by propagation over the resulting poly-tree or by an equivalent variable elimination procedure [10].

If the instantiated separator set does not cut all loops we can add additional nodes from the clique until we get a full loop-cutset. If the resulting loop-cutset (containing the separator variables) has size c_s, clique's processing is time exponential in c_s only and not in the full size of the clique.

In summary, given a join-tree decomposition, we can choose a loop-cutset of a clique C_i that is a minimal subset of variables, which together with its parent separator-set constitute a loop-cutset of the subnetwork defined over C_i. We conclude:

Theorem 4: Lets n be the number of variables in a belief network. Given a constant $s \leq n$, let T_s be a clique-tree whose separator size has size s or less, and let c_s^* be the maximum size of a minimal cycle-cutset in any subgraph defined by the cliques in T_s. Then belief assessment and *mpe* can be computed in space $O(n \cdot exp(s))$ and in time $O(n \cdot exp(c_s^*))$, and $c_s^* \geq s$, while c_s^* is smaller than the clique size.

Proof: Since computation in each clique is done by the cycle-cutset conditioning, its time is exponentially bounded by c_s^*, the maximal cycle-cutset over all the cliques of T_s, denoted c_s^*. The space complexity remains exponential in the maximum separator s. Since for every clique, the loop-cutset we select contains its parent separator, then clearly $c_s^* > s$. $\Box$

Algorithm STC is presented in Figure 4. We conclude with an example that demonstrate the time-space tradeoff when using cycle-cutset in each clique.

Example 3: Considering the join-trees in Figure 3, if we apply the cycle-cutset scheme inside each subnetwork defined by each clique, we get no improvement in the bound for T_0 because the largest loop-cutset size in each cluster is 3 since it always exceeds the largest separator. Remember that once a loop-cutset is instantiated, processing the simplified network by propagation or by any efficient method is also $O(k^2)$. However, when using the secondary tree T_1, we can reduce the time bound from $O(k^5)$ to $O(k^4)$ with only $O(k^2)$ space because the cutset size of the largest subgraph restricted to $\{G, D, E, F, H\}$, is 2; in this case the separator $\{G, D\}$ is already a loop-cutset and therefore when applying conditioning to this subnetwork the overall time complexity is now $O(k^4)$. When applying conditioning to the clusters in T_2, we get a time bound of $O(k^5)$ with just $O(k)$ space because, the loop-cutset of the subnetwork over $\{B, C, D, G, E, F, H\}$ has three nodes only $\{B, G, E\}$. In summary, the dominating tradeoffs (when considering only the exponents) are between an algorithm based on T_1 that requires $O(k^4)$ time and quadratic space and an algorithm based on T_2 that requires $O(k^5)$ time and linear space.

3 Constraint Networks

Constraint networks have proven successful in modeling mundane cognitive tasks such as vision, language comprehension, default reasoning, and abduction, as well as in applications such as scheduling, design, diagnosis, and temporal and spatial reasoning. In general, constraint satisfaction tasks are computationally intractable.

Algorithm space-based join-tree clustering (STC(s))
Input: A belief network (G, P), where G is a DAG and $P = \{P_1, ..., P_n\}$, a space parameter s.
Output: the belief of X_1 given evidence e.

1. Generate a join-tree clustering of G, call it T_0.
2. Generate the secondary join-tree by combining any two adjacent cliques whose separator is strictly larger than s. Let $C_1, ...C_t$ be the cliques in the resulting secondary join-tree. Place each P_i and each observation in one clique that contains its arguments.
3. Impose directionality on the secondary join-tree, Let $d = C_1, ...C_t$ be a breadth-first ordering of the rooted clique-tree, let $S_{p(i)}$ and $C_{p(i)}$ be the parent separator and the parent clique of C_i, respectively.
4. From $i \leftarrow t$ downto 1 do
5. (Processing clique C_i with cycle-cutset):
   Find a subset of variables $I_i \subseteq C_i$ s.t. $I_i \cup S_{p(i)}$ is a loop-cutset of the subgraph of G restricted to nodes C_i.
   Let $\lambda_1, \lambda_2, ..., \lambda_j$ be the functions in clique C_i,
   - For any observation $X_j = x_j$ in C_i assign x_j to each function.
   - For every assignment $\bar{x}$ of $S_{p(i)}$ do,
     $\lambda_p(\bar{x}) \leftarrow 0$.
     For every assignment $\bar{y}$ of I_i do, $(U_i = C_i - I_i - S_{p(i)})$
     - Using the cycle-cutset scheme compute:
       $\lambda(\bar{x}, \bar{y}) \leftarrow \sum_{\{u_i | S_{p(i)}=\bar{x}, I_i=\bar{y}\}} \Pi_{i=1}^{j} \lambda_i$.
     - $\lambda_p(\bar{x}) \leftarrow \lambda_p(\bar{x}) + \lambda(\bar{x}, \bar{y})$

     Put λ_p in parent clique $C_{p(i)}$.
6. **Return** (processing root-clique, C_1), $Bel(x_1) = \alpha \sum_{u_1} \Pi_i \lambda_i$
   α is a normalizing constant.

Figure 4: Algorithm Space-based join-tree clustering

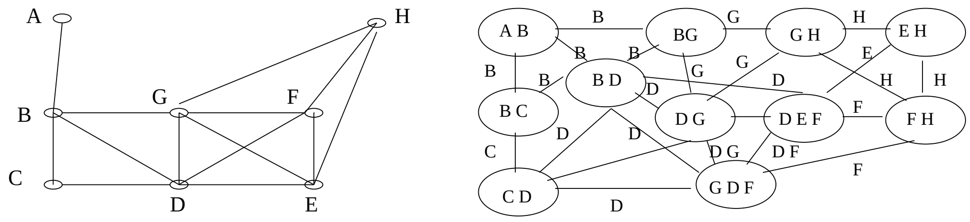

Figure 5: Primal (a) and dual (b) constraint graphs

.

Definition 5 [Constraint network]: A *constraint network* consists of a finite set of variables $X = \{X_1, \ldots, X_n\}$, each associated with a domain of discrete values, $D_1, \ldots, D_n$ and a set of constraints, $\{C_1, \ldots, C_t\}$. A *constraint* is a relation, defined on some subset of variables, whose tuples are all the compatible value assignments. A constraint C_i has two parts: (1) the subset of variables $S_i = \{X_{i_1}, \ldots, X_{i_{j(i)}}\}$, on which the constraint is defined, called its *scope*, and (2) a *relation*, rel_i, defined over $S_i : rel_i \subseteq D_{i_1} \times \cdots \times D_{i_{j(i)}}$. The *scheme* of a constraint network is the set of scopes on which constraints are defined. An assignment of a unique domain value to each member of some subset of variables is called an *instantiation.* A consistent instantiation of *all* the variables that does not violate any constraint is called a *solution.* Typical queries associated with constraint networks are to determine whether a solution exists and to find one or all solutions. A *primal constraint graph* represents variables by nodes and associates an arc with any two nodes residing in the same constraint. A *dual constraint graph* represents each constraint subset by a node and associates a labeled arc with any two nodes whose constraint subsets share variables. The arcs are labeled by the shared variables.

Figure 5a and b present the primal and the dual graphs of a constraint problem.

Tree clustering for constraint networks is similar to join-tree clustering for probabilistic networks. In fact, the structuring part is identical. Once the join-tree structure is determined, each constraint is placed in a clique (or a cluster) that contains its scope and then each clustered subproblem can be solved independently. The *time and space complexity* of tree-clustering is governed by the time and space required to generate the relations of each clique in the join-tree which is exponential in the clique's size, and therefore in the problem's induced width w^* [14, 9]. Since the graph in Figure 5(a) is identical to the graph in Figure 1(b), it possesses the same clique-tree embeddings.

Refining the clustering method for constraint networks can be done just as we did for probabilistic networks, thus tree-clustering in constraint networks obeys similar time and space complexities. The directional version of join-tree clustering for finding a solution to a set of constraints is given in Figure 6. We can show:

Theorem 5: [Time-space of tree-clustering][14]: Given a constraint problem over n variables whose constraint graph can be embedded in a clique-tree having tree width r and separator size s, the time complexity of tree-clustering for deciding consistency and for finding one solution is $O(n \cdot exp(r))$ and its space complexity is $O(n \cdot exp(s))$. The time complexity for generating all solutions is $O(n \cdot exp(r) + |solutions|)$, also requiring $O(n \cdot exp(s))$ memory. □

When the space required by clustering is beyond the available resources, tree-clustering can be coerced to yield smaller separators and larger subproblems, as we have seen earlier for processing belief networks. This leads to a conclusion similar to Theorem 3.

Theorem 6: Given a constraint network over n variables whose constraint graph can be embedded in a primary clique-tree having separator sizes $s_0, s_1, ..., s_k$, whose corresponding maximal clique sizes in the secondary join-trees are $r_0, r_1, ..., r_k$, then deciding consistency and finding a solution can be accomplished using any one of the time and space complexity bounds $O(n \cdot exp(r_i))$ and $O(n \cdot exp(s_i))$, respectively.

Proof: Analogous to Theorem 3. □ □

Finally, similar to belief networks, any linear-space method can replace backtracking for solving each of the subproblems defined by the cliques. One possibility is to use the cycle-cutset scheme. The cycle-cutset method for constraint networks (like in belief networks) enumerates the possible solutions to a set of cycle-cutset variables and, for each consistent cutset assignment, solves the restricted tree-like problem in polynomial time. Thus, the overall time complexity is bounded by $O(n \cdot k^{c+2})$, where c is the cutset size, k is the domain size, and n is the number of variables [8]. Therefore,

Algorithm directional tree-clustering for CSPs
Input: A set of constraints $R_1, ..., R_l$ over $X = \{X_1, ..., X_n\}$, having scopes $S_1, ..., S_l$ respectively, and its constraint graph G.
Output: A solution to the constraint problem.

1. Generate a join-tree clustering of G, identified by its cliques $C_1, ...C_t$. Place each R_i in one clique that contains its scope.
2. Impose directionality on the join-tree, namely create a rooted directed tree whose root is any clique . Let $d = C_1, ...C_l$ be a breadth-first ordering of the rooted clique-tree, let $S_{p(i)}$ and $C_{p(i)}$ be the parent separator and the parent clique of C_i, respectively.
3. From $i \leftarrow l$ downto 1 do
4. (Processing clique C_i):
   Let $R_1, R_2, ..., R_j$ be the constraints in clique C_i, let U_i be the set of variables in clique C_i.
   - Solve the subproblem in C_i and call the set of solutions ρ_i. Project this set of solutions on the parent separator. Let $\rho_{S_{p(i)}}$ be the projected relation. $\rho_{S_{p(i)}} \leftarrow \prod_{S_{p(i)}} \Join_{k=1}^{j} R_k$
     Put $\rho_{S_{p(i)}}$ in parent clique $C_{p(i)}$.
5. **Return** generate a solution in a backtrack-free manner going from the root clique towards the leaves.

Figure 6: Algorithm directional join-tree clustering for constraints

Theorem 7: Let G be a constraint graph over n variables and let T be a join-tree with separator size s or less. Let c_s be the largest minimal cycle-cutset[3] in any subproblem in T. Then the problem can be solved in space $O(n \cdot exp(s))$ and in time $O(n \cdot exp(c_s + 2))$, where $c_s \geq s$. $\Box$

Example 4: Applying the cycle-cutset method to each subproblem in T_0, T_1, T_2 (see Figure 3), yields the same time-space tradeoffs as for the belief network case.

A special case of Theorem 7, observed before in [13, 17], occurs when the graph is decomposed into non-separable components (i.e., when the separator size equals 1).

Corollary 1: *If G has a decomposition to non-separable components such that the size of the maximal cutsets in each component is bounded by c, then the problem can be solved in $O(n \cdot exp(c))$ time, using linear space.* $\Box$

4 Optimization Tasks

Clustering and conditioning are applicable also to optimization tasks defined over probabilistic and deterministic networks. An optimization task is defined relative to a real-valued criterion or cost function associated with every instantiation. In the context of constraint networks, the task is to find a consistent instantiation having maximum cost. Applications include diagnosis and scheduling problems. In the context of probabilistic networks, the criterion function denotes a utility or a value function, and the task is to find an assignment to a subset of decision variables that maximize the expected criterion function. Applications include planning and decision making under uncertainty. If the criterion function is decomposable, its structure can be augmented onto the corresponding graph to subsequently be exploited by either tree-clustering or conditioning.

Definition 6 [Decomposable criterion function [3, 25]]: A criterion function over a set X of n variables $X_1, ..., X_n$ having domains of values $D_1, ..., D_n$ is additively decomposable relative to a scheme $Q_1, ..., Q_t$ where $Q_i \subseteq X$ iff

$$f(x) = \sum_{i \in T} f_i(x_{Q_i}),$$

[3]As before, the cycle-cutset contains the separator set

where $T = \{1, ..., t\}$ is a set of indices denoting the subsets of variables $\{Q_i\}$ and x is an instantiation of all the variables. The functions f_i are the components of the criterion function and are specified, in general, by stored tables.

Definition 7 [Constraint optimization, augmented graph] Given a constraint network over a set of n variables $X = X_1, ..., X_n$ and a set of constraints $C_1, ..., C_t$ having scopes $S_1, ..., S_t$, and given a criterion function f decomposable into $\{f_1, ..., f_l\}$ over $Q_1, ..., Q_l$, the constraint optimization problem is to find a consistent assignment $x = (x_1, ..., x_n)$ such that the criterion function $f = \sum_i f_i$, is maximized. The *augmented constraint graph* contains a node for each variable and an arc connecting any two variables that appear either in the same scope of a constraint or in the same functional component of the criterion function.

Since constraint optimization can be performed in linear time when the augmented constraint graph is a tree, both join-tree clustering and cutset-conditioning can extend the method to non-tree structures [32] in the usual manner. We can conclude:

Theorem 8: [32]*[Time-space of constraint optimization]:* Given a constraint optimization problem over n variables whose augmented constraint graph has a cycle-cutset of size c, and whose augmented graph can be embedded in a clique-tree having tree width r and separator size s, the time complexity of finding an optimal consistent solution using tree-clustering is $O(n \cdot exp(r))$ and the space complexity $O(n \cdot exp(s))$. The time complexity for finding a consistent optimal solution using the cycle-cutset conditioning is $O(n \cdot exp(c))$ while its space complexity is linear. □

In a similar manner, the structure of the criterion function can augment the moral graph when computing the maximum expected utility (MEU) of some decisions in a general influence diagram [20]. For more details see [12].

Once we have established the graph that guides tree-clustering and conditioning, the same principle of trading space for time becomes applicable and will yield a collection of parameterized algorithms governed by the primary and secondary clique-trees and cycle-cutsets of the augmented graphs as we have seen before. For completeness sake we restate the full theorem:

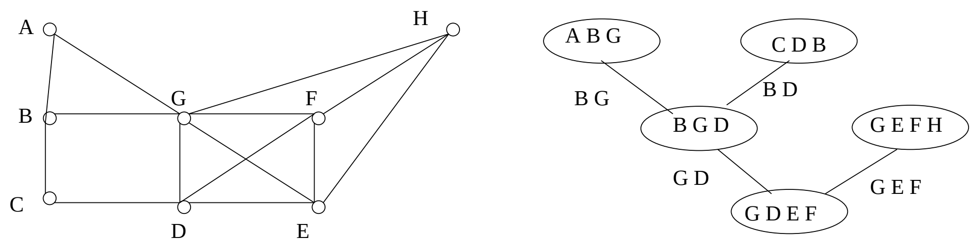

Figure 7: An augmented moral graph for the utility function $f(a,b,c,d,e,f,g.h) = a \cdot g + c^2 + 5d \cdot e \cdot f$

Theorem 9: Given a constraint network over n variables and given an additively decomposable criterion function f, if the augmented constraint graph relative to the criterion function can be embedded in a clique-tree having separator sizes $s_0, s_1, ...s_k$, and corresponding maximal clique sizes $r_0, r_1, ..., r_k$ and corresponding maximal minimal cutset sizes $c_0, c_1, ..., c_k$, then finding an optimal solution can be accomplished using any one of the following bounds on the time and space: if a brute-force approach is used for processing each subproblem the bounds are $O(n \cdot exp(r_i))$ time and $O(n \cdot exp(s_i))$ space. If cycle-cutset conditioning is used for each cluster, the bounds are $O(n \cdot exp(c_i))$ time and $O(n \cdot exp(s_i))$ space, where $c_i \geq s_i$. $\Box$

Example 5: Consider the following criterion function defined over the constraint network in Figure 5

$$u(a,b,c,d,e,f,g,h) = a \cdot g + c^2 + 5d \cdot e \cdot f.$$

Here the augmented graph will have one additional arc connecting nodes A and G (see Figure 7(a)), resulting in a primary clique-tree embedding in Figure 7(b) that differs from the tree in Figure 3(a). As a result one has to consider the clique ABG instead of the original clique AB. Thus, applying join-tree clustering to the primary tree yield time complexity $O(exp(4))$ and space complexity $O(k^3)$. If only binary functions can be recorded we will need to combine clique $(GDEF)$ with $(GEFH)$ yielding a clique of size 5. Using cycle-cutset conditioning, this results in time complexity of $O(k^4)$ as well, while using $O(k^2)$ space, only. If this space requirement is too heavy we need to solve the whole problem as one cluster using cycle-cutset conditioning which, in this case, requires $O(k^5)$ time and linear space.

5 Empirical framework and results

The motivation for the experiments is twofold. One, to analyze the structural parameters of clustering and cutset on real-life instances. Two, to gain further understanding of how space/time tradeoff can be exploited to alleviate space bottlenecks. We analyzed empirically benchmark combinatorial circuits, widely used in the fault diagnosis and testing community [6]. (See Table 1.) The experiments allow us to assess in advance the complexity of diagnosis and abduction tasks on those circuits, and to determine the appropriate combination of tree clustering and cycle cutset methods to perform those tasks for each instance. None of the circuits are trees and they all have considerable fanout nodes as shown in the schematic diagram of circuit c432 in Fig. 8.

Table 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

A directed acyclic graph (DAG), is computed for each circuit. The graph includes a node for each variable in the circuit. For every gate in the circuit, the graph has an edge directed from each gate's input to the gate's output. The nodes with no parents (children) in the DAG are the primary inputs (outputs) of the circuit. The primal graph for each circuit is then computed as the moral graph for the corresponding DAG. Table 2 gives the number of

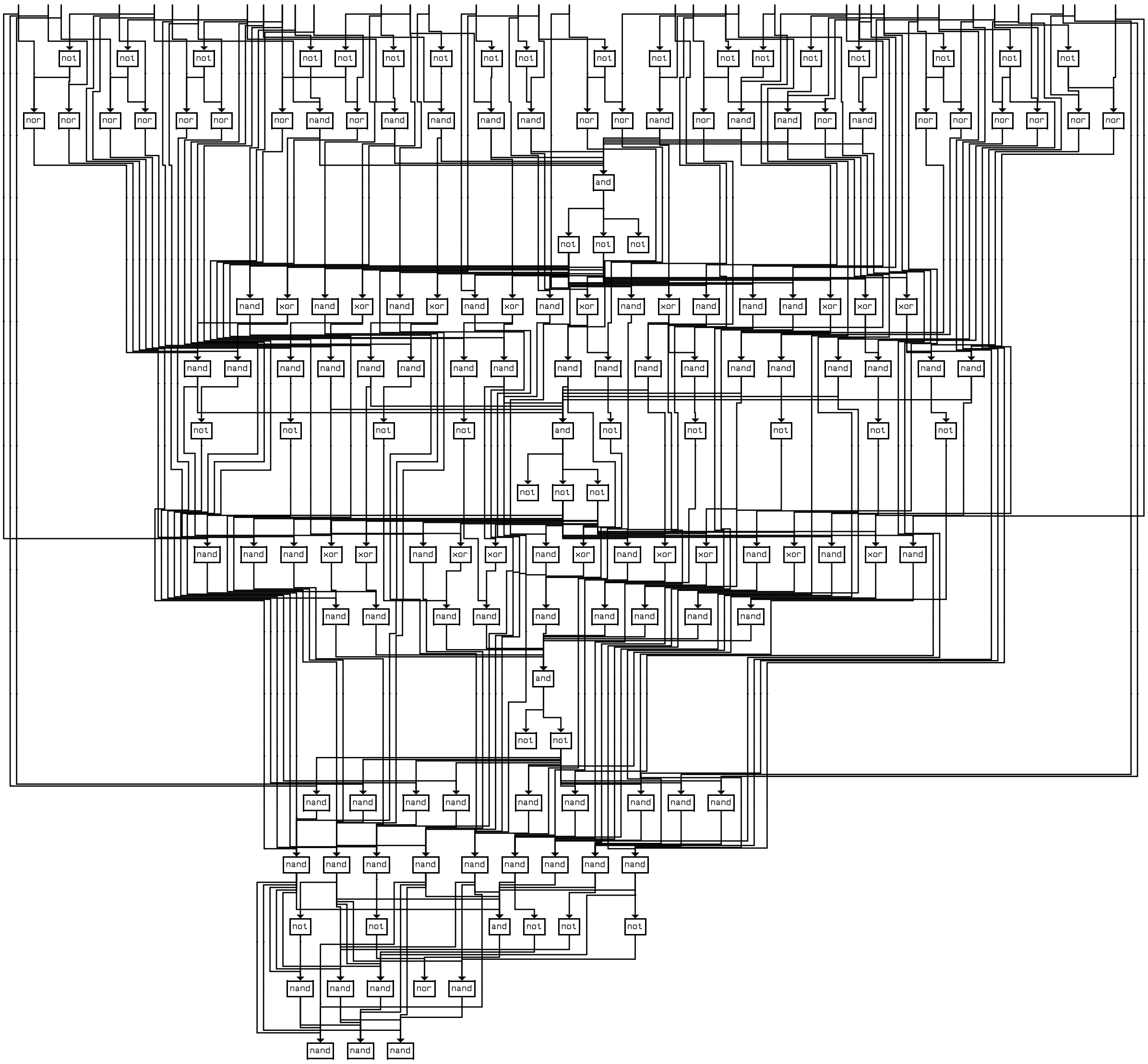

Figure 8: Schematic of circuit c432: 36 inputs 7 outputs and 160 components.

Table 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

nodes and edges of the primal graph for each circuit.

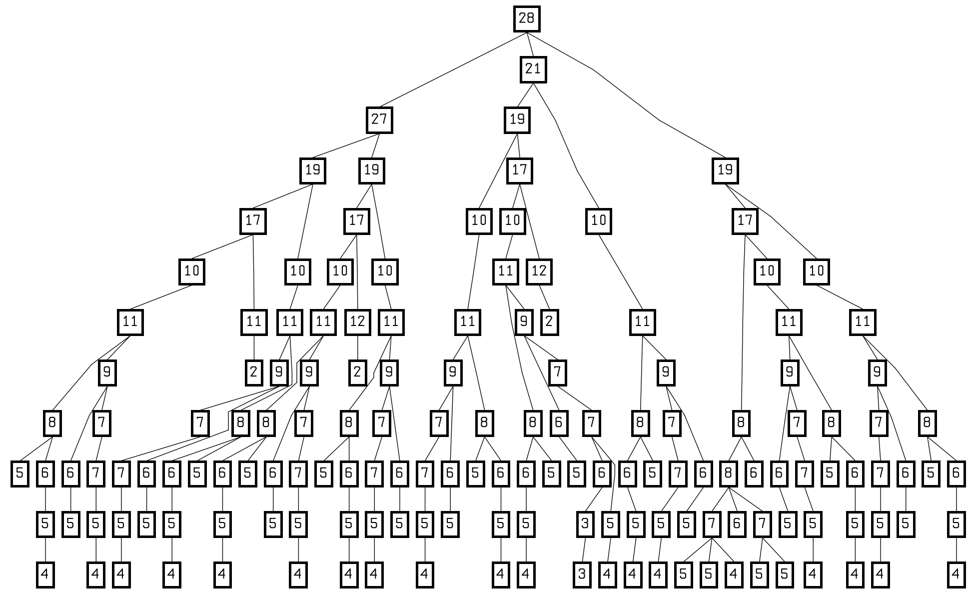

Figure 9: Primary join tree (157 cliques) for circuit c432 (196 variables); the maximum separator size is 23.

Tree clustering is performed on the primal graphs as usual: by selecting an ordering for the nodes, then triangulating the graph and identifying its maximum cliques. There are many possible heuristics for ordering the nodes with the aim of obtaining a join tree with small cliques.

Four ordering heuristics were considered: (1) causal ordering, (2) maximum-cardinality, (3) minimum-width, and (4) minimum-degree. The max-cardinality ordering is computed from first to last by picking the first node arbitrarily and then repeatedly selecting the unordered node that is adjacent to the maximum number of already ordered nodes. The min-width ordering is computed from last to first by repeatedly selecting the node having the least number of neighbors in the graph, removing the node and its incident edges from the graph, and continuing until the graph is empty. The min-degree ordering [5] is exactly like min-width except that we connect neighbors of selected nodes, and the causal ordering is just a topological sort of the directed graph for the circuit.

Tree clustering was implemented using each of the four orderings on each of the benchmark circuits of Table 1 and we observed that the min-degree ordering was by far the best, yielding the smallest cliques sizes and separators. Our evaluation of the performance of the orderings is consistent with the results in [23]. Therefore, we report here the results only relative to the *min-degree ordering*. For results on the other orderings see [12].

5.1 Results: 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. In this section we present the results on two circuits c432 and c3540, having 196 and 1719 variables, respectively. Results on other circuits are summarized in [12].

Figure 9 and 10 present information on the primary join trees. For example, Figure 9 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.

Figures 11 and 12 provide additional details showing the frequencies of cliques sizes, separator sizes and cutsets sizes for both circuits. Those figures (and all the corresponding figures in [12])

show that the structural parameters are skewed with the vast majority of the parameters having values below the midpoint (the point dividing the range of values from smallest to largest).

We see in Figure 11 that the number of cliques is 157 and the cliques sizes

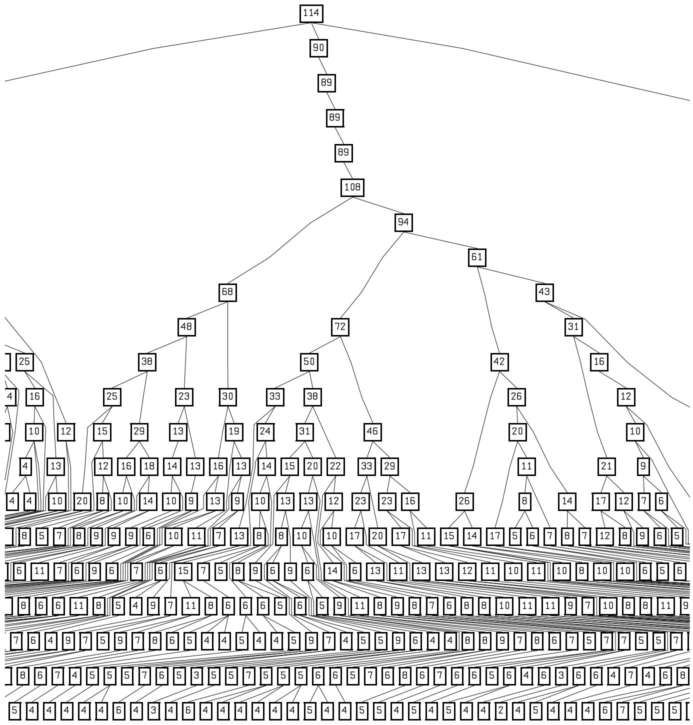

Figure 10: Part of primary join tree (1419 cliques) for circuit c3540 (1719 variables) showing the descendants of the root node down to the leaves; the maximum separator size is 89.

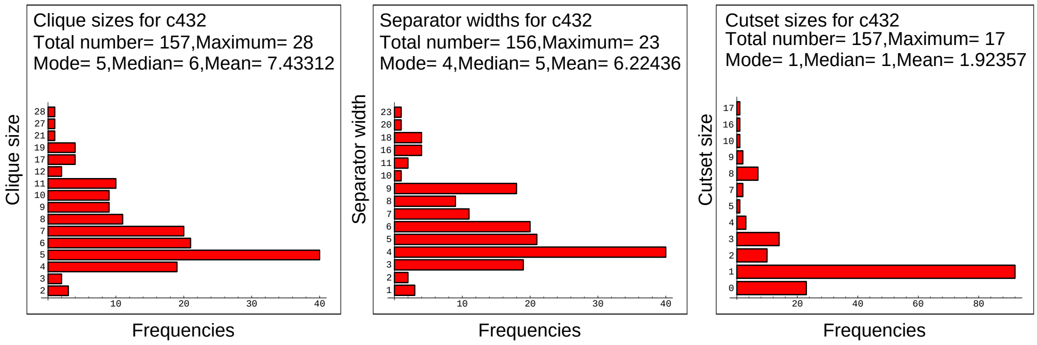

Figure 11: Histograms of the cliques sizes, the separator sizes and the cutsets sizes of the primary join tree for circuit c432 (196 variables)

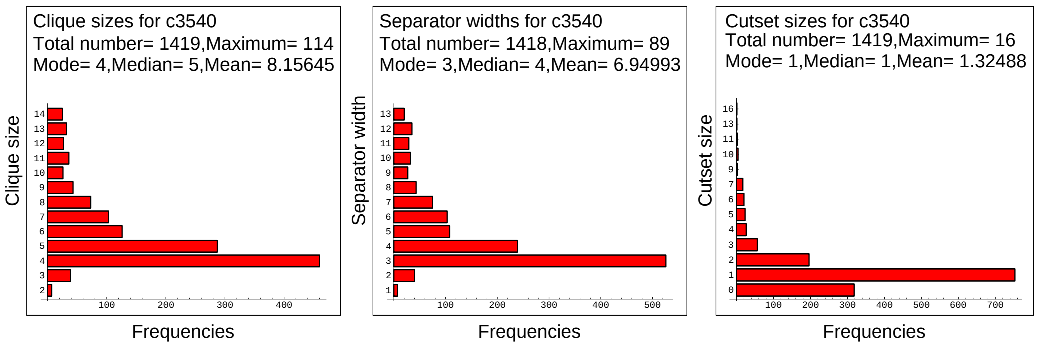

Figure 12: Histograms of the cliques sizes (0.9th quantile range), the separator sizes (0.9th quantile range) and the cutsets sizes of the primary join tree for circuit c3540 (1719 variables).

are in the range from 2 to 28. The mode is 5, the median is 6 and the mean is 7.433. 40 cliques out of the total 157 have size 5, and only 23 out of 157 have size greater than 9. The separator sizes are in the range from 1 to 23. The mode is 4, the median is 5 and the mean is 6.224. Out of the total 156 separator sizes, 40 have size 4 and only 13 have sizes greater than 10. The cutsets sizes are in the range from 0 to 17. The mode is 1, the median is 1 and the mean is 1.923. Out of 157 cliques, 23 have cutset size 0. This means that the projection of the primal graph on each of those 23 cliques is already acyclic.

The corresponding figures for C3540 can be read from Figure 12. Figure 12 shows the 0.9th quantile distribution of the separator sizes. Like the cliques, 90% of the separator sizes are small (between 1 and 13) and the remaining 10% span a broad range of values (from 14 to 89). For the cutsets sizes we note that 318 cutsets out of total 1419 have size 0, namely the projection on each of those 318 cliques is already acyclic. We also note that 753 out of 1419 cliques have singleton cutsets. Only 47 out of 1419 cutsets have sizes greater than 5.

5.2 Results: Hybrid Clustering + Conditioning

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 the trade off scheme proposed in this paper.

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.

We estimate the space/time bounds for each circuit based on the graph parameters observed using our tree decomposition scheme. Figure 13 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 maximum between the separator size and the cutset size; $max(s_i, c_i)$.

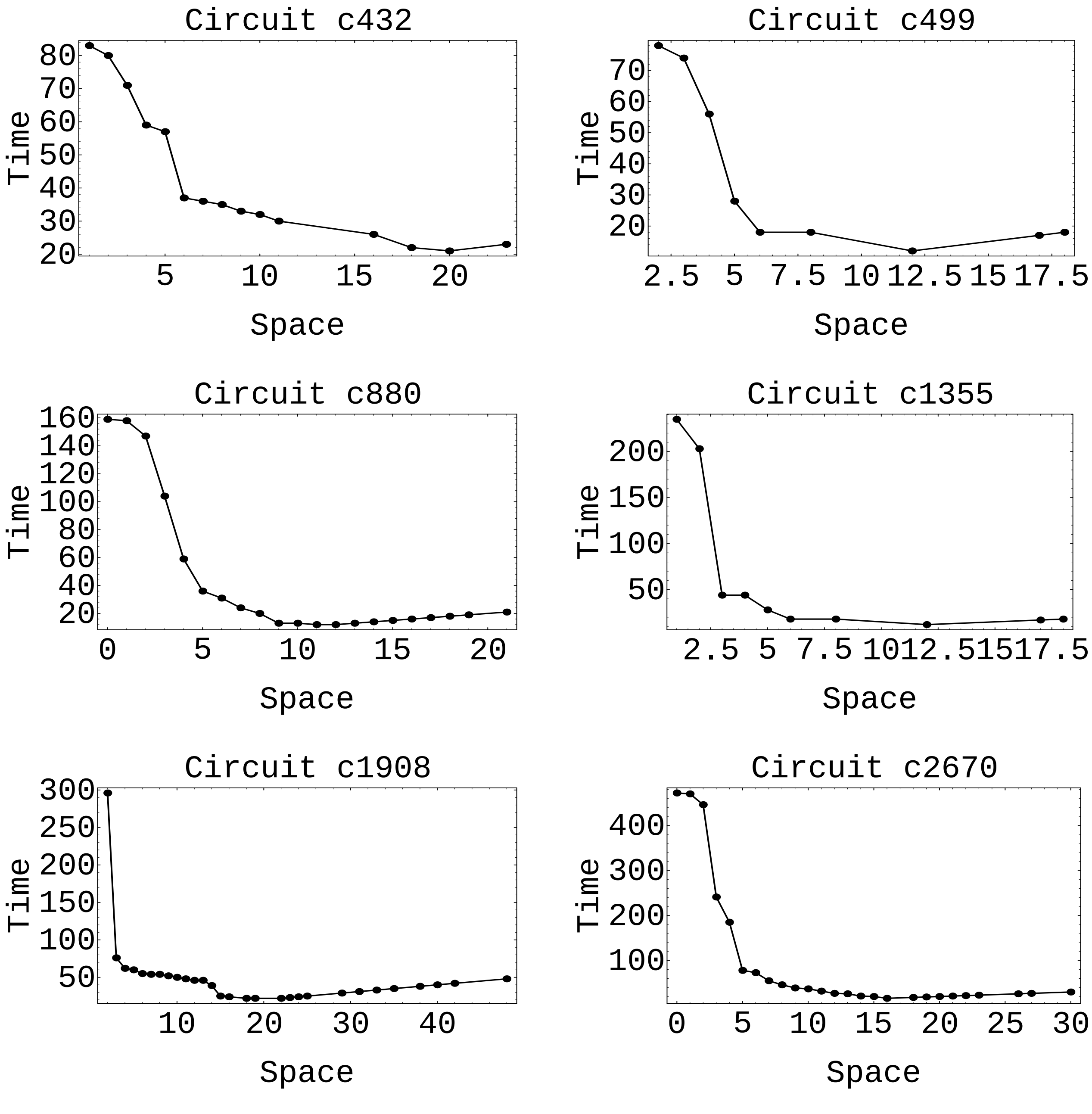

Figure 13: Time/Space tradeoff for c432 (196 variables), c499 (243 variables), c880 (443 variables), c1355 (587 variables), c1908 (913 variables) and c2670 (1426 variables). Time is measured by the maximum of the separator size and the cutset size and space by the maximum separator size.

Each chart in Figure 13 can be used to select the algorithm from the spectrum of conditioning+clustering algorithms of our tree decomposition scheme that best meets a given time-space specification. They show the gradual effect of lowering space on the time required by a corresponding clustering+conditioning algorithm. For example circuit c432 (Figure 13) 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 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.

Note that the charts in Figure 13 all display a "knee" phenomenon in the time-space tradeoff where time increases only slightly for a wide range of space reduction beyond which further reduction in space causes significant rise in the time bound. Note also that the time estimate shown in Figure 13 displays a small dip before it rises with the decrease in available space. For example for circuit c432 we note a dip when the space is decreased from 23 to 20.

Figures 14 and 15 display the structure of secondary join trees for c432. The primary join tree for the circuit is shown in Figure 9. The secondary trees are indexed by the separator sizes of the primary tree which range from 1 to 23 (Figure 11). 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.

6 Related work

The cycle-cutset scheme for probabilistic inference was introduced by Pearl [29] and for constraint networks by Dechter [8]. It was further improved and extended for probabilistic reasoning by [30, 7, 15].

In subsequent years the cycle-cutset scheme was recognized as a special case of *conditioning*, namely, value assignments to a subset of variables creates subproblems that can be solved by any means. While the cycle-cutset scheme requires that the conditioning set will be large enough so that the

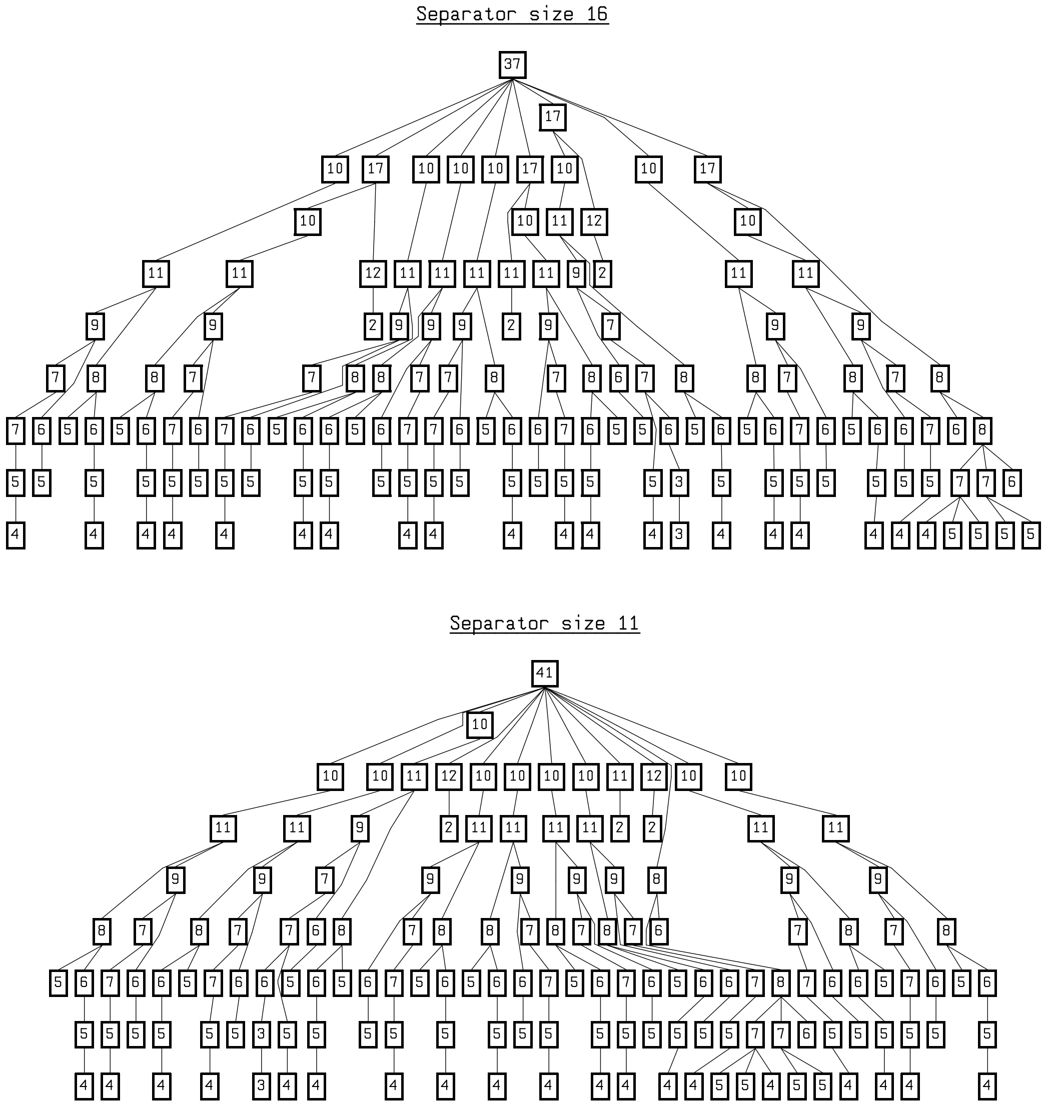

Figure 14: Secondary trees for c432 with separator sizes 16 and 11

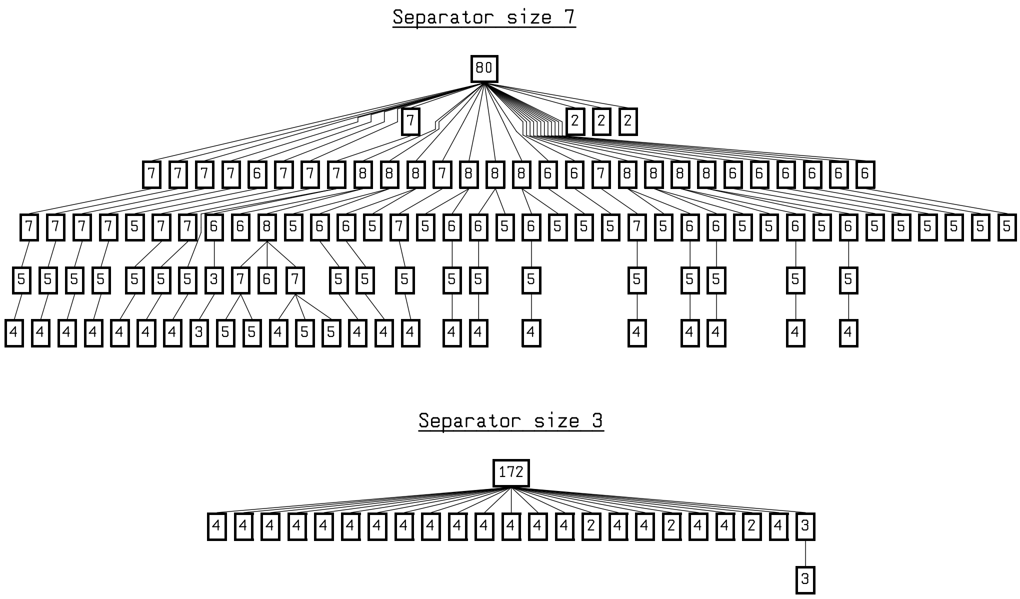

Figure 15: Secondary trees for c432 with separator sizes 7 and 3.

resulting subproblem is singly-connected, any size of conditioning set can be used, yielding simplified problems that can be solved by tree-clustering or by any other method. This idea of extending the combination of conditioning and tree-clustering beyond the cycle-cutset scheme appears in the work of Jegou [21] for constraint networks and in the works of [31, 19] for probabilistic networks. In the work of Jegou various heuristic are presented, aiming at creating a hybrid algorithm having improved time performance. In [31], the issue of reducing the space of tree-clustering by combination with conditioning is also briefly addressed. The latter paper includes an alternative proof to the (worst-case) time superiority of tree-clustering over the cycle-cutset method. In [19] the idea is applied to the pathfinder system, when the conditioning set is restricted to the set of diseases.

Finally in [33, 34], a scheme for combining conditioning and variable elimination for propositional theories is outlined and analyzed. It is shown that although the worst-case time guarantee of an hybrid cannot be superior to tree-clustering (nor to a variable elimination scheme), for some problem classes a hybrid algorithm can have a better time performance than both pure clustering and pure search. The work in this paper provides an alternative hybrid scheme between conditioning and elimination.

7 Summary and Conclusions

Problem solving methods can be viewed as hybrids of two main principles: inference and search. Tree-clustering is an example of an inference algorithm while the cycle-cutset is an example of a search method. Tree clustering algorithms are time and space exponential in the size of their cliques while search algorithms are time exponential but require only linear memory. In this paper we developed a structure-based hybrid scheme that uses tree-clustering and cycle-cutset conditioning 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

Specifically, we have shown that constraint processing and belief network processing obey a structure-based time-space tradeoff, that allows tailoring a combination of tree-clustering and cycle-cutset conditioning to certain time and space requirements. As well, the same tradeoff is obeyed by optimization problems when augmenting the graph by arcs reflecting the structure of the criterion function. Our analysis presents a spectrum of algorithms that allows a rich time-space performance balance, applicable across a variety of tasks.

The structural parameters of interest are: (1) the size of cliques in a join tree, 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.

To assess the applicability of our scheme to real-life domains, we studied the structural parameters of 11 benchmark circuits widely used in the fault diagnosis and testing community [6]. We observed that the join-trees of the circuits all shared the unexpected property that while few cliques are distinctly large, the majority of cliques sizes are relatively small. Also, the distributions of all the structural parameters are skewed. This observation has an important practical implication. Although the primary join tree obtained by tree-clustering may require too much space, a major portion of the tree can be solved without any space problem.

Our analysis should be qualified, however. All the results present worst-case guarantees of the corresponding algorithm. It is still not clear that the bounds are tight nor that they correlate with average case performance. This analysis should be extended in the future to include implementing and testing of the involved algorithms, a major effort outside the scope of this paper.

References

[1] S. Arnborg and A. Proskourowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete and Applied Mathematics*, 23:11–24, 1989.

[2] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.

[3] F. Bacchus and P. van Run. Dynamic variable ordering in csps. In *Principles and Practice of Constraints Programming (CP'95)*, Cassis, France, 1995. Available as Lecture Notes on CS, vol 976, pp 258 – 277, 1995.

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

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

[6] Brglez and Fujiwara. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in fortran. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1996.

[7] A Darwiche. Conditioning algorithms for exact and approximate inference in causal networks. In *Uncertainty in Artificial Intelligence (UAI-95)*, pages 99–107, 1995.

[8] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[9] R. Dechter. Constraints networks. In *Encyclopedia of Artificial Intelligence, 2nd Ed.*, pages 276–285, 1992.

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

[11] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[12] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. Technical report, 1999.

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

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

[15] F. J. Diez. Local conditioning in bayesian networks. *Artificial Intelligence*, 87:1–20, 1996.

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

[17] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[18] H. Geffner and J. Pearl. An improved constraint propagation algorithm for diagnosis. In *Proceedings of IJCAI-87*, pages 1105–1111, Milan, Italy, 1987.

[19] G. F. Cooper H. J. Suermondt and D. E. Heckerman. A combination of cutset conditioning with clique-tree propagation in the path-finder system. In *Uncertainty in Artificial Intelligence (UAI'91)*, pages 245–253, 1991.

[20] R. A. Howard and J. E. Matheson. *Influence diagrams*. 1984.

[21] P Jegou. Cyclic clustering: a compromise between tree-clustering and the cycle-cutset method for improving search efficiency. In *European Conference on AI (ECAI-90)*, pages 369–371, Stockholm, 1990.

[22] F.V. Jensen, S.L Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

[23] U. Kjæaerulff. Triangulation of graph-based algorithms giving small total state space. In *Technical Report 90-09, Department of Mathematics and computer Science*.

[24] U. Kjæaerulff. Nested junction trees. In *Uncertainty in Artificial Intelligence (UAI'97)*, pages 294–301, 1997.

[25] D. H. Krantz, R.D. Luce, P. Suppes, and A. Tversky. Foundations of measurements, academic press. 1976.

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

[27] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25, 1985.

[28] J. Pearl. Fusion propagation and structuring in belief networks. *Artificial Intelligence*, 29(3):241–248, 1986.

[29] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[30] M. A. Peot and R. D. Shachter. Fusion and proagation with multiple observations in belief networks. *Artificial Intelligence*, pages 299–318, 1992.

[31] S.K. Anderson R. D. Shachter and P. Solovitz. Global conditioning for probabilistic inference in belief networks. In *Uncertainty in Artificial Intelligence (UAI'94)*, pages 514–522, 1994.

[32] A. Dechter R. Dechter and J. Pearl. Optimization in constraint networks. In *Influence Diagrams, Belief Nets and Decision Analysis*, pages 411–425. John Wiley & Sons, 1990.

[33] I. Rish and R. Dechter. To guess or to think? hybrid algorithms for sat. In *Principles of Constraint Programming (CP-96)*, pages 555–556, 1996.

[34] I. Rish and R. Dechter. Resolution vs. sat: two approaches to sat. *Journal of Approximate Reasoning (to appear)*, 2000.

[35] D. G. Corneil S. A. Arnborg and A. Proskourowski. Complexity of finding embeddings in a k-tree. *SIAM Journal of Discrete Mathematics.*, 8:277–284, 1987.

[36] T. Schmidt and P.P. Shenoy. Some improvements to the shenoy-shafer and hugin architecture for computing marginals. *Artificial Intelligence*, 102:323–333, 1998.

[37] G. Shafer. Probabilistic expert systems. *Society for indutrial and Applied Mathematics, Philadelphia, PA*, 1996.

[38] G. R. Shafer and P.P. Shenoy. Probability propagation. *Anals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[39] K. Shoiket and D. Geiger. A proctical algorithm for finding optimal triangulations. In *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 185–190, 1997.

[40] S. Srinivas. A probabilistic approach to hierarchical model-based diagnosis. In *Working Notes of the Fifth International Workshop on Principles of Diagnosis*, pages 305–311, New Paltz, NY, USA, 1994.