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

3 June 1999

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 optimization tasks. By analyzing the problem structure the user can select from a spectrum of hybrid 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. Inference algorithms are time and space exponential in the size of the relationships they record while search algorithms are time exponential but require only linear memory. In this paper we develop a hybrid scheme that uses inference (tree-clustering) and search (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 the application and the available resources.

In general, 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 that can then be solved by a specialized efficient tree-solving algorithm [1,2]. 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 minus one. 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 [3]. 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 [4] in 1974 and later reafirmed in [5]. 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, $r \leq c$, where c is the minimal cycle-cutset and r is the tree-width [4]. 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 width*, which is defined as the size of the maximum subset of variables shared by adjacent subproblems in the join-tree. Our investigation employs the separator width 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 parametrized 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 [6–8]. We analyze 11 benchmark combinatorial circuits widely used in the fault diagnosis and testing community [9] (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.

In order to demonstrate our claims we use a directional variant of tree-clutering that is query-based which simplify the exposition. However, the approach is applicable to the general version of tree-clustering.

Section 2 gives definitions and preliminaries and introduces the time-space tradeoff ideas using belief networks. Sections 3 and 4 extends these ideas to constraint networks and to optimization problems. Section 5 describes the empirical framework and Section 6 presents the results. Section 7 discusses related works and section 8 gives our conclusion.

The paper assumes familiarity with the basic concepts of tree-clustering and cycle-cutset conditioning and provides only breif 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 occurrance 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 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).

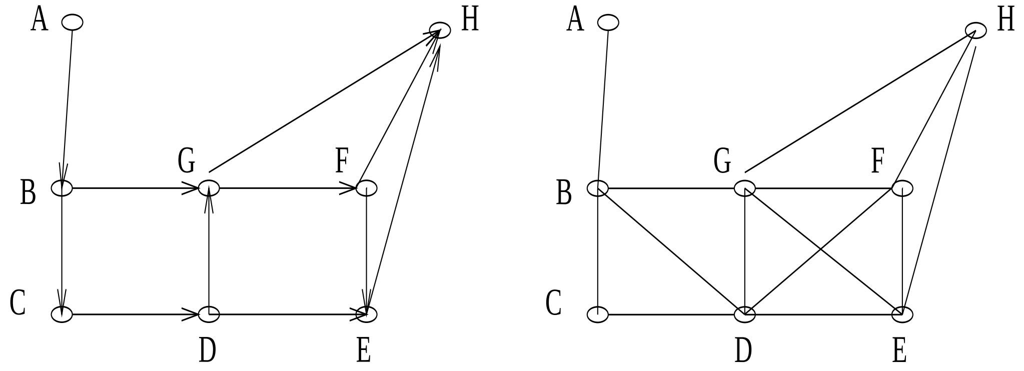

Fig. 1. (a) A belief network and (b) its moral graph

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 [10].

Example 4 Figure 1 shows a belief network's acyclic graph and its associated moral graph. The induced-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.

Two of the most common tasks over belief networks are, to determine posterior beliefs and to find the most probable explanation (mpe), given a set of observations, or evidence. It is well known that such tasks can be answered effectively for singly-connected polytrees by a belief propagation algorithm [11]. This algorithm can be extended to multiply-connected networks by either tree-clustering or loop-cutset conditioning [11].

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 by graph information only, 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 with whom it shares a maximal number of variables. This clique is called its parent clique. 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). For more information on structuring a join-tree, induced-width and join-tree clustering see [11,12,3,13].

A tighter bound on the space complexity of tree-clustering may be obtained using the *separator width*. The separator width of a join-tree is the maximal size of the intersections between any two cliques, and the *separator width of a graph* is the minimal separator width over all the graph's join-trees [12,14].

Tailoring the join-tree clustering scheme to answer only a given query is another way to save time. Algorithm *Directional tree clustering (DTC)*, presented in Figure 2, is a query-based variant of join-tree clustering. It improves tree-clustering by 1) recording functions on separators only and 2) by restricting to query-based answers only. Clearly, each of this modification can be incorporated independently. Once the structuring part of the join-tree is determined, (the cliques are connected in a tree-structure and each clique has a parent clique and a parent separator), then each CPT is placed in one clique that contains its arguments. For example, given the join-tree structure in Figure 3a, the CPT $P(B|A)$ is placed in clique AB, $P(C|B)$ and $P(D|C)$ are placed in clique BCD, clique BDG contains $P(G|B,D)$, Clique $GDEF$ contains $P(E|D,F)$ and $P(F|G)$, and finally, clique $GEFH$ contains $P(H|G,F,E)$.

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.

(i) 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.

(ii) 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.

(iii) From $i \leftarrow t$ downto 1 do

(iv) (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 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 $C_{p(i)}$.

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

Fig. 2. Algorithm directional join-tree clustering

Subsequently, 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] .

Algorithm directional tree-clustering (DTC), is presented in Figure 2 for the belief updating task. It can be adapted for the task of finding the most probable explanation (mpe), by replacing the summation operation by maximization. The algorithm tightens the bound on space complexity using its separator width. This modification to its space management can be applied to any variant of tree-clustering which is not necessarily query-based. In summary,

Theorem 5 (Time-space of join-tree clustering) Given a belief network whose moral graph can be embedded in a clique-tree having induced width r and separator width s, the time complexity for determining beliefs and the mpe by a join-tree clustering (e.g., by DTC) is $O(n \cdot exp(r))$ while its space

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

complexity is $O(n \cdot exp(s))$. $\Box$

Clearly $s \leq r$. Note that since in the example of Figure1 the separator width 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 width is much smaller than the induced-width.

2.1.3 Cycle-cutset conditioning

Belief networks may be processed also by cutset conditioning [11]. A subset of nodes is called a *cycle-cutset* of an undirected graph 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 [11,15]. 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 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 [11]. 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 [11] showed that weights computation is not more costly than enumerating all the conditioned beliefs.

This scheme was later simplified by Peot and Shachter [15]. 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. In summary,

Theorem 6 ([11,15]) Given a probabilistic network having a loop-cutset of

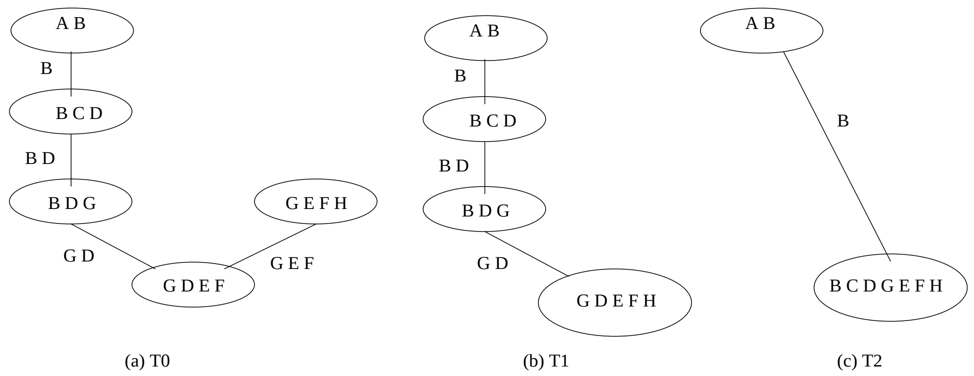

Fig. 3. A tree-decomposition with separators equal to (a) 3, (b) 2, and (c) 1

size c, belief updating and mpe can be computed in time $O(exp(c+2))$ [2] and in linear space. □

2.2 Trading Space for Time

Assume now that we have a problem whose join-tree has induced width r and separator width 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 7 (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

Theorem 8 Given a join-tree T, 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 randing over all the sequence of secondary join-trees), respectively.

[2] the "2" in the exponent comes from the fact that belief updating on trees is linear in the size of the CPTs which are at least $O(c^2)$.

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.

Example 9 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 [16,17]. 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. However, finding w_s^* may be hard as well. 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 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 [18].

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 that constitute a loop-cutset of the subnetwork defined over C_i. We conclude:

Theorem 10 Given a constant $s \leq n$, let T_s be a clique-tree whose separator width 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 $c_s^* > s$.

Next we give two examples. The first demonstrates the time-space tradeoff when using cycle-cutset in each clique. The second demonstrates in details the mechanism of processing each clique by the cycle-cutset method.

Example 11 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 also 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 $O(exp(2))$ space because the cutset size of the largest subgraph restricted to $\{G, D, E, F, H\}$, is 2; in this case the separator $\{C, D\}$ 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 $O(k)$ space.

Example 12 We conclude this section by demonstrating through our example in Figure 1, the mechanics of processing a subnetwork by (1) a brute-force methods and (2) by loop-cutset conditioning. We will use join-tree T_2 and

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.

(i) Generate a join-tree clustering of G, call it T_0.

(ii) 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.

(iii) 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.

(iv) From $i \leftarrow t$ downto 1 do

(v) (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 $C_{p(i)}$.

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

Fig. 4. Algorithm Space-based join-tree clustering

process cluster $\{B, C, D, E, F, G, H\}$. Processing this cluster amounts to computing the marginal distribution over the separator B, namely for every value $b\prime$ of B (we annotate constant by primes):

$$P(b') = \sum_{d,g,e,c,f,h} (P(g|b', d)P(c|b')P(d|c)P(f|g)P(e|d, f)P(h|g, f, e).$$

Migrating the components as far to the left as possible, we get.

$$P(b') = \sum_{d} \sum_{g} P(g|b', d) \sum_{e} \sum_{c} P(c|b')P(d|c) \sum_{f} P(f|g)P(e|d, f) \sum_{h} P(h|g, f, e). \quad (1)$$

Brute-force computation of expression 1 will accumulate the needed sums along the search tree of all possible instantiations to the variables. In this case we expand the probability tree in a forward manner assigning values one by one to all variables, and computing the tuple's probability. The time complexity is exponential in 6 (the size of the probability search tree) and requires linear space only, since the sums can be accumulated while traversing the tree in a depth-first manner.

If we use the loop-cutset method within this cluster, we can condition on G, E (in addition to to the separator B). This makes the resulting problem a poly-tree. Mechanically, it means that we will expand the probability tree forward for variables G, E, and for every assignment $g\prime, e\prime$ (B can also be viewed as if it is assigned because it is the queried variable) we will use the poly-tree algorithm (or an equivalent variable elimination procedure which treats b, g, e as constant.) We demonstrate the backwards computation (denoting constants by primes) using variable elimination that is equivalent to belief propagations on the poly-tree resulting from instantiating g, e, b. We need to compute

$$P(b', g', e') = \sum_d \sum_g P(g|b', d) \sum_e \sum_c P(c|b')P(d|c)$$
$$\sum_f P(f|g')P(e'|d, f) \sum_h P(h|g', f, e'). \quad (2)$$

We compute $h_H(f) = \sum_h P(h|g', f, e')$. This takes time exponential in 2 and space exponential in 1. Subsequently, compute $h_F(d) = \sum_f P(f|g')P(e'|d, f)H_h(f)$ which takes time exponential in 1 and constant space. Finally, sum the result over variable C: $h_C(d) = \sum_c P(c|b')P(d|c)H_F(d)$ in time exponential in 1 and constant space. So far we spent time exponential in 2 and space exponential in 1 at the most. Since we have to repeat this for every value of d, e, b the overall time will be exponential in 5 while space is bounded by $O(k)$.

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.

Definition 13 (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 $S_i : rel_i \subseteq D_{i_1} \times \cdots \times D_{i_{j(i)}}$.

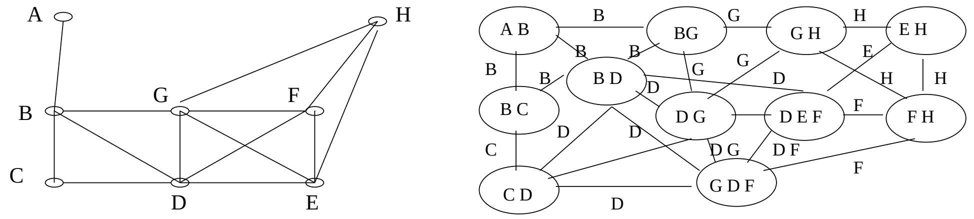

Fig. 5. Primal (a) and dual (b) constraint graphs

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.

Definition 14 (Constraint graphs) Two graphical representations of a constraint network are its primal constraint graph and its dual constraint graph. 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.

Example 15 Figure 5 depicts the primal and the dual representations of a network having variables A, B, C, D, E, F, G, H whose constraints are defined on the subsets $\{(A,B),(B,C),(B,D),$ $(C,D),(D,G),(G,E),(B,G),$ $(D,E,F),(G,D,F),(G,H),(E,H)(F,H)\}$.

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. In other words, the set of constraints in a clique can be replaced by its set of solutions; a new constraint whose scope is the clique's variables. 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^* [12,3].

Example 16 Since the graph in Figure 5(a) is identical to the graph in Figure 1(b), it possesses the same clique-tree embeddings. Namely, the maximal cliques of the chordal graph are $\{(A,B),(B,C,D),(B,D,G),$
$(G,D,E,F),(H,G,F,E)\}$ and a join-tree is given in Figure 3(a). The schemes of each clique's subproblem are:
$C_{AB} = \{(A,B)\}$, $C_{BCD} = \{(B,C),(C,D)\}$,

$C_{BDG} = \{(B,D),(B,G),(G,D)\}$,
$C_{GDEF} = \{(G,D),(D,E),(E,F),(G,F),(D,F)\}$
$C_{GEFH} = \{(E,F),(G,F),(G,H),(F,H),(E,H)\}$.
As in the probabilistic case, a brute-force application of tree-clustering to this problem is time and space exponential in 4.

The ideas underlying tree-clustering and conditioning in constraint networks are like those for belief networks. In particular, by refining the clustering method for constraint networks just as we did for probabilistic networks, it is easy to see that tree-clustering in constraint networks obeys similar time and space complexities. Specifically, deciding the consistency of a tree of binary constraints, can be done by directional arc-consistency (also called pair-wise consistency) along some directed rooted tree. If the empty relation is not generated, finding one solution can be done in a backtrack-free manner from root to leaves [10]. Applying directional arc-consistency to a join-tree whose nodes are the cliques and whose values are the solutions of the subproblems defined by the clique, is likewise valid. The operation of solving each subproblem in the clique-tree and the operation of pair-wise consistency can be interleaved. First a rooted clique-tree is created. Then cliques are processed from leaves to root. The solutions of leaf-cliques are generated first (perhaps by a backtracking search algorithm) and their projection are recorded on their parent's separator. Subsequently, each parent clique processes its subproblem augmented with the recorded constraints, projects the solutions on its own parent clique, and so on. Therefore, constraints may be recorded only on their parent separator. The time complexity of this modified algorithm is exponential in the tree width, while its space complexity is exponentially bounded only by the maximal separator between subproblems. The directional version of join-tree clustering for finding a solution to a set of constraints is given in Figure 6.

We conclude:

Theorem 17 (Time-space of tree-clustering [12]) Given a constraint problem whose constraint graph can be embedded in a clique-tree having tree width r and separator width 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. $\Box$

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 8.

Theorem 18 Given a constraint network whose constraint graph can be embedded in a primary clique-tree having separator sizes $s_0, s_1, ..., s_n$, whose cor-

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.

(i) 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.
(ii) 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.
(iii) From $i \leftarrow$ downto 1 do
(iv) (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. $\rho_{S_{p(i)}} \leftarrow \prod_{S_{p(i)}} \bowtie_{k=1}^{j} R_k$
  Put $\rho_{S_{p(i)}}$ in parent clique $C_{p(i)}$.
(v) **Return** generate a solution in a backtrack-free manner going from the root clique towards the leaves.

Fig. 6. Algorithm directional join-tree clustering for constraints

responding maximal clique sizes in the secondary join-trees are $r_0, r_1, ..., r_n$, 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. □

Similarly 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 exponential in the size of the cycle-cutset of the graph [19]. More precisely, it can be shown that the cycle-cutset method 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 [19]. Fortunately, enumerating all the cycle-cutsets assignments can be accomplished in linear space using a backtracking algorithm.

Theorem 19 Let G be a constraint graph 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 $c_s \geq s$.

Proof. Since the maximum separator size is s, then, from Theorem 17, tree-clustering requires $O(n \cdot exp(s))$ space. Since the cycle-cutset's size in each cluster is bounded by c_s, the time complexity is exponentially bounded by c_s.

Example 20 Applying the cycle-cutset method to each subproblem in T_0, T_1, T_2 shows (see Figure 3), as before, that the best alternatives are an algorithm having $O(k^4)$ time and quadratic space, (using T_1, since $\{G, E\}$ is a cycle-cutset of the subgraph restricted to $\{G, D, E, F, H\}$ having size 2), and an algorithm having $O(k^5)$ time and $O(k)$ space only (using T_2, since the cycle-cutset size of the whole problem is 3).

A special case of Theorem 3, observed before in [10,20], is that when the graph is decomposed into non-separable components (i.e., when the separator size equals 1).

Corollary 21 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. $\square$

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

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

augmented onto the corresponding graph (constraint graph or moral graph) to subsequently be exploited by either tree-clustering or conditioning.

Definition 22 (Decomposable criterion function [21]) 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

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

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 23 (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 [22] in the usual manner. We can conclude:

Theorem 24 (Time-space of constraint optimization [22]) Given a constraint optimization problem whose augmented constraint graph can be embedded in a clique-tree having tree width r and separator width s and a cycle-cutset size c, 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. $\Box$

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 [23]. An *influence diagram* is a belief network having decision variables as well as an additively decomposable utility function.

Definition 25 (Finding the MEU) Given a belief network BN defined on a set of variables X, and a real-valued utility function $u(x)$ that is additively decomposable relative to $Q_1, ..., Q_t$, $Q_i \subseteq X$, and given a subset of decision variables $D = \{D_1, ... D_k\}$ that are root variables in the directed acyclic graph of the BN, the MEU task is to find an assignment $\bar{d^o_k} = (d^o{}_1, ..., d^o{}_k)$ such that

$$(\bar{d^o_k}) = argmax_{\bar{d}_k} \sum_{x_{k+1},...,x_n} \Pi_{i=1}^k P(x_i | x_{pa(X_i)}, \bar{d}_k) u(x).$$

The utility-augmented graph of an influence diagram is its moral graph with some additional edges: any two nodes appearing in the same component of the utility function are connected as well.

A linear-time propagation algorithm can compute the MEU whenever the utility-augmented moral graph of the network is a tree [24]. Consequently, by exploiting the augmented moral graph, we can extend this propagation algorithm to general influence diagrams. The two approaches that extend this propagation algorithm to multiply-connected networks, cycle-cutset conditioning and join-tree clustering, are applicable here as well [11,13,23]. It was also shown that elimination algorithms are similar to tree-clustering methods [12]. In summary:

Theorem 26 (Time-space of finding the MEU) Given a belief network having a subset of decision variables, and given an additively decomposable utility function whose augmented moral graph can be embedded in a clique-tree having tree width r and separator width s and a cycle-cutset size c, the time complexity of computing the MEU using tree-clustering is $O(n \cdot exp(r))$ and the space complexity is $O(n \cdot exp(s))$. The time complexity for finding a MEU using cycle-cutset conditioning is $O(n \cdot exp(c))$ while the space complexity is linear. □

Once we have established the graph that guides tree-clustering and conditioning for either constraint optimization or for finding the MEU, the same principle of trading space for time becomes applicable and will yield a collection of parametrized algorithms governed by the primary and secondary clique-trees and cycle-cutsets of the augmented graphs as we have seen before.

The following theorem summarizes the time and space tradeoffs associated with optimization tasks.

Theorem 27 Given a constraint network (resp., a belief network) and given an additively decomposable criterion function f, if the augmented constraint graph (resp., moral graph) relative to the criterion function can be embedded in a clique-tree having separator sizes $s_0, s_1, ... s_n$, and corresponding maximal clique sizes $r_0, r_1, ..., r_n$ and corresponding maximal minimal cutset sizes $c_0, c_1, ..., c_n$, then finding an optimal solution (resp., finding the maximum expected criterion value) 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

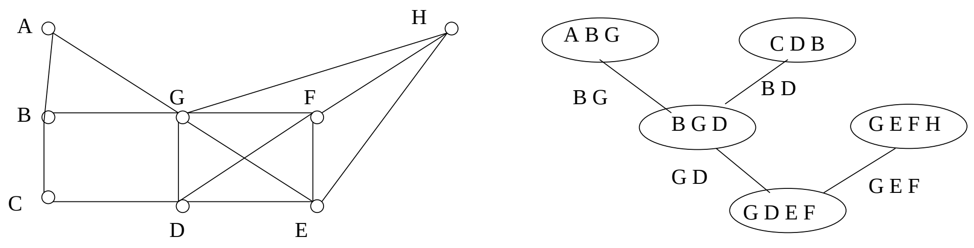

Fig. 7. An augmented moral graph for the utility function $f(a,b,c,d,e,f,g.h) = a \cdot g + c^2 + 5d \cdot e \cdot f$

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 $c_i \geq s_i$. □

In particular, if the criterion function has singleton components, then the complexity bounds, when using tree-clustering for optimization, are identical to the performance bounds for constraint satisfaction or to the performance bound for finding the posterior probabilities.

Example 28 Consider the following criterion function defined over the belief network in Figure 1

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

Assume that we want to compute the expected utility. Here the augmented moral 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

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. With that motivation in mind, we analyzed empirically benchmark combinatorial circuits, widely used in the fault diagnosis and testing community [9]. (See table 1.) The experiments allow us to assess in advance the worst-case 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

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

A causal graph, namely 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 nodes and edges of the primal graph for each circuit.

Tree clustering is performed on the primal graphs by first 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.

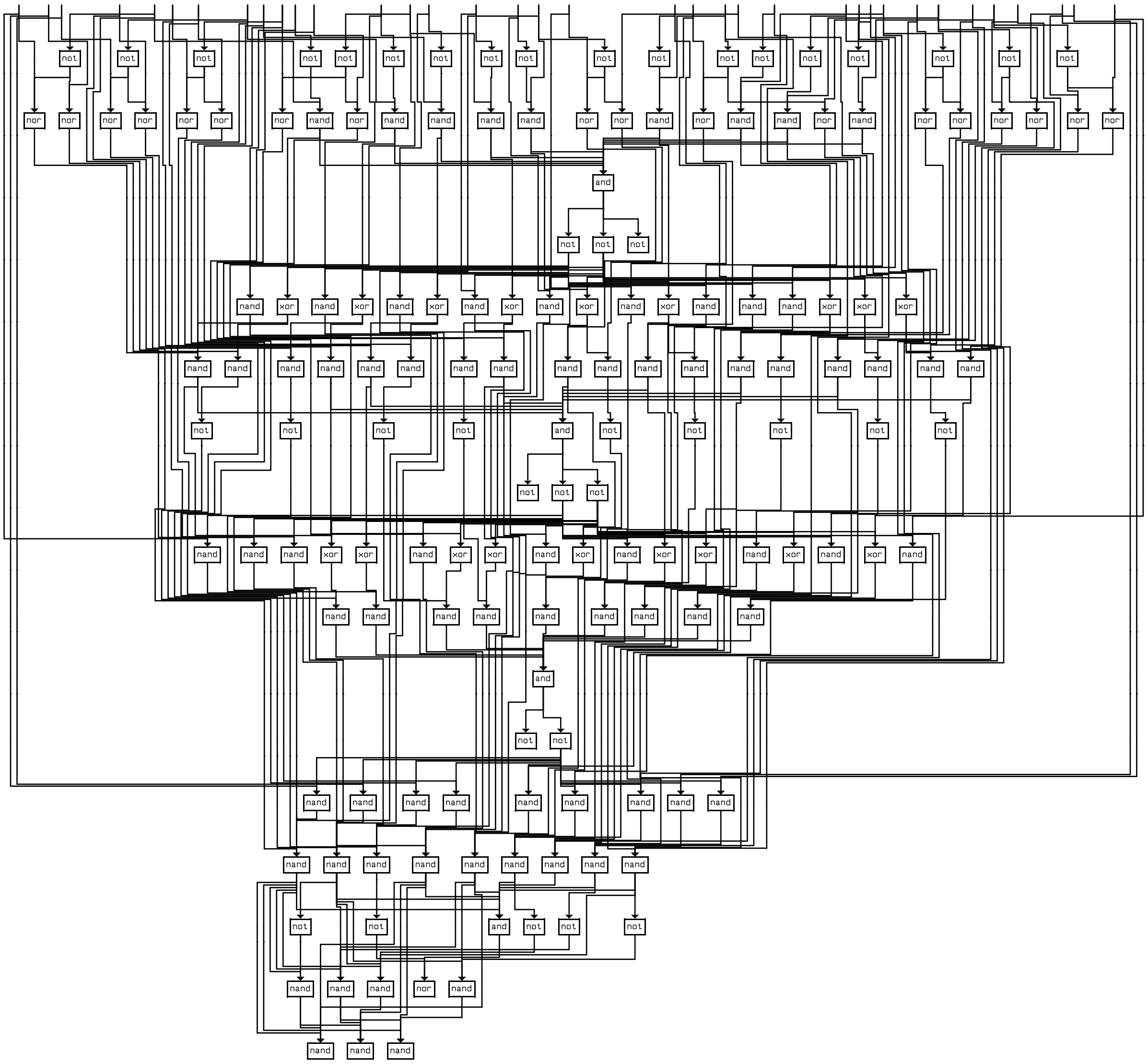

Fig. 8. Schematic of circuit c432: 36 inputs 7 outputs and 160 components.

As a side effect of our experiments we observed a dramatic effect of ordering heuristics on the resulting primary join tree. (See Appendix B). We report here the results relative to the *min-degree ordering*, which we found to be superior. The min-degree ordering heuristic was proposed in the context of non-serial dynamic programming [4, page 55]. According to this ordering, nodes are ordered from last to first by repeatedly selecting a node with minimum degree (minimum number of neighbors) from the current graph, connecting the node's neighbors then removing the node from the graph, and continuing until the graph is empty.

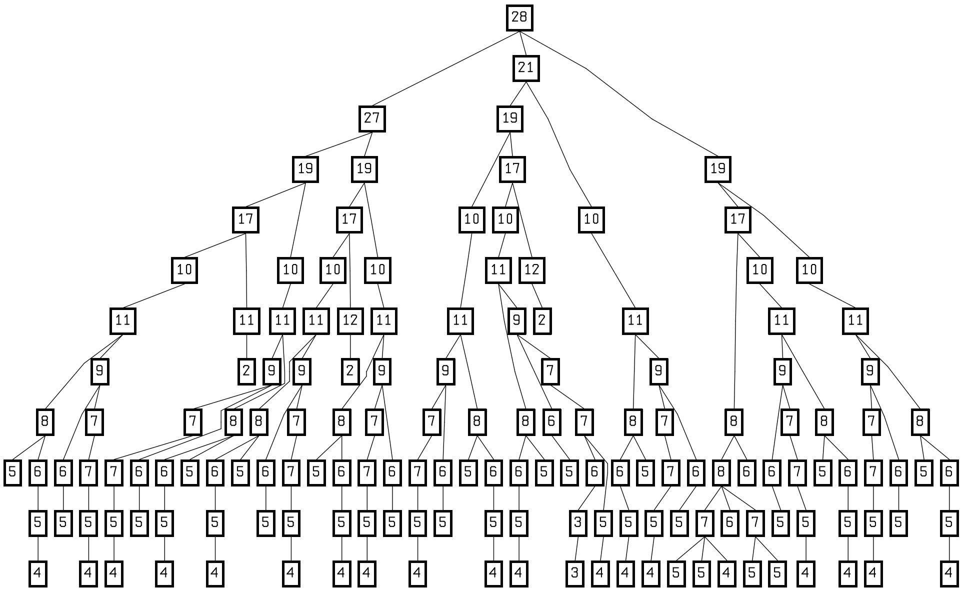

Fig. 9. Primary join tree (157 cliques) for circuit c432 (196 variables); the maximum separator width is 23.

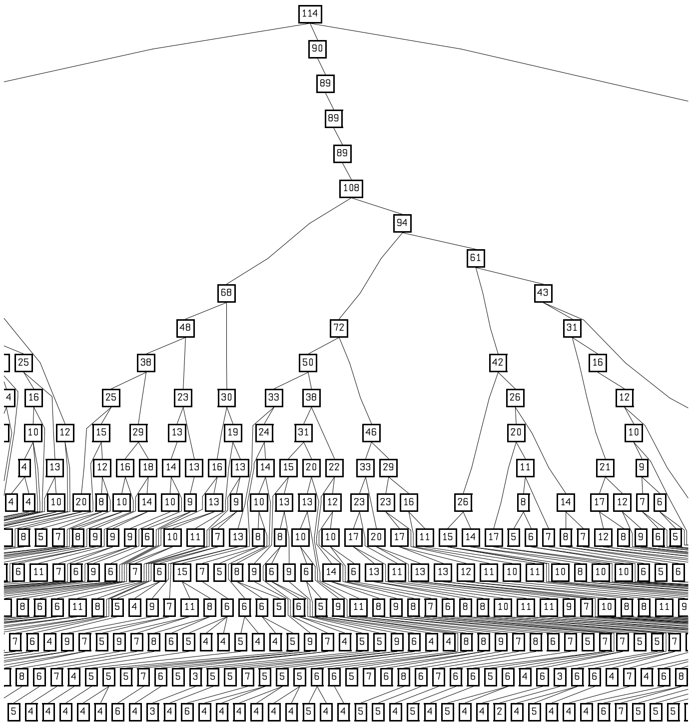

Fig. 10. Part of pimary join tree (1419 cliques) for circuit c3540 (1719 variables) showing the descendants of the root node down to the leaves; the maximum separator width is 89.

6 Empirical Results

6.1 Pure Clustering; 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 width. 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 Appendix A and C.

Figure 9 and 10 present information on the primary join trees. 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 stricly less than 10. The leaves have sizes ranging from 2 to 6. Similar observations can be made for the primary join tree of the larger circuit c3540 shown in Fig. 10.

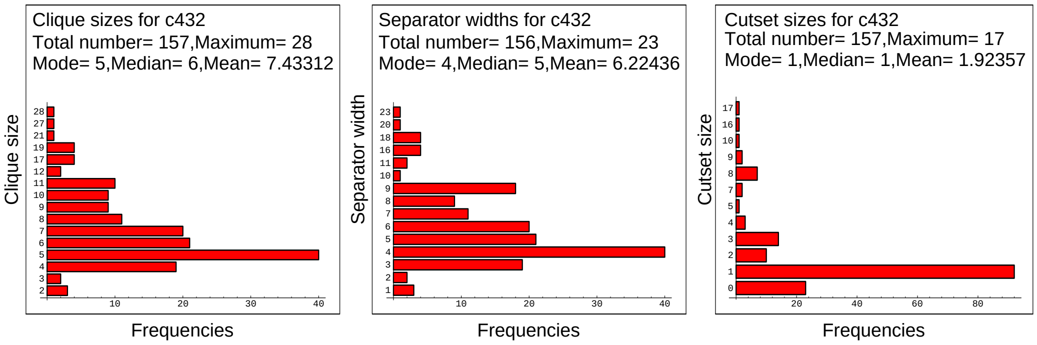

Fig. 11. Histograms of the cliques sizes, the separator widths and the cutsets sizes of the primary join tree for circuit c432 (196 variables)

Figures 11 and 12 provide additional details showing the frequencies of cliques sizes, separator widths and cutsets sizes for both circuits. Those figures (and all the corresponding figures in Appendix A) 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 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 widths 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 widths, 40 have size 4 and only 13 have sizes greater than 10. The

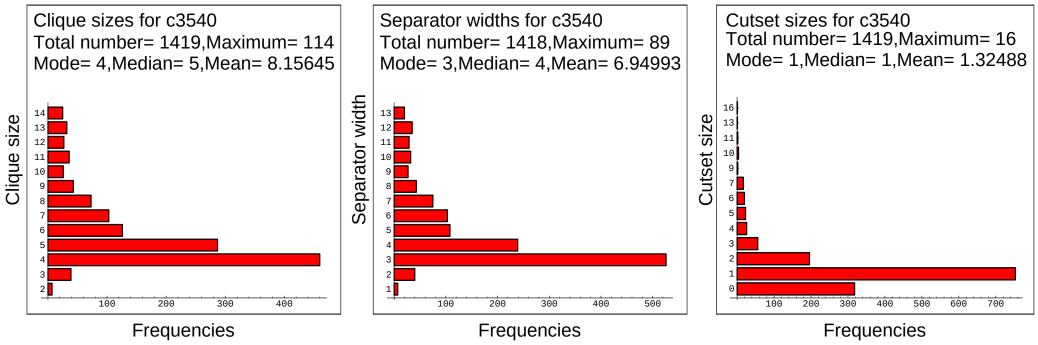

Fig. 12. Histograms of the cliques sizes (0.9th quantile range), the separator widths (0.9th quantile range) and the cutsets sizes of the primary join tree for circuit c3540 (1719 variables).

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.

We now analyze the data for circuit c3540. (See Fig. 12.) The figure shows the distribution of cliques sizes in the 0.9th quantile range. The number of cliques is 1419 and the cliques sizes range from 2 to 114. The mode is 4, the median is 5 and the mean is 8.1564. Out of 1419 cliques, 400 have size 4. Although the maximum clique size is 114, the majority of cliques (1284 out of 1419) have sizes between 2 and 14 and only few (135 out of 1419) have sizes ranging from 15 to the maximum 114. Figure 12 also shows the 0.9th quantile distribution of the separator widths. Like the cliques, 90% of the separator widths 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.

6.2 Hybrid Clustering + Conditioning; Secondary Join Trees

Although most cliques and separators are small, some will 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 providing bounds for time versus space for each circuit. Each point in the chart corrseponds to a specific secondary join tree decomposition T_i and has the space complexity measured by the separator width, s_i, and the time complexity by the maximum between the separator width and the cutset size; $max(s_i, c_i)$. 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. Each chart shows the gradual effect of lowering the available space on the time required by a corresponding clustering+conditioning algorithm. For example circuit c432 (Figure 13) shows the separator width (space) which is initially 23 (for the primary join tree) gradually reduced down to 1 in a series of secondary trees. The figure shows that reducing the separator width to meet the space available for a conditioning+clustering algorithm increases the worst-case time complexity of the algorithm. The time complexity increases becasue of the larger clusters contained in the secondary join tree and the increase in the size of the cutset for those clusters. It is interesting to note that the charts in Figure 13 all display a "knee" phenomenon in the time-space tradeoff where time increases only slightly for a broad range of space reduction beyond which further reduction in space causes significant rise in the time bound. We also note 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. This can be explained as follows. As mentioned earlier we measure the worst-case time complexity as the maximum between the separator width and the cutset size. This is so since time complexity always exceeds space complexity. As the space measured by the separator width decreases the secondary join trees will contain larger clusters and their cutset size will also increase. However, the rate of increase of the cutset sizes does not grow linearly with the decrease in the separator width. The results shown in Figure 13 indicate that for small decrease in separator width (space) the cutset size will increase by an amount smaller than the amount of decrease in the separator width. This explains the dip in the time complexity. However, beyond that dip further decrease in the separator width (space) will cause an increase in the cutset size that dominates the decrease in the separator width and the rate of time increase becomes staggering as the algorithms approach the lowest bound of linear space.

Figures 14-16 display the structure of the join trees for c432 for separator widths ranging from 23 down to 3. As the separator decreases the maximum clique size increases, and both the size and the depth of the tree decrease. Like

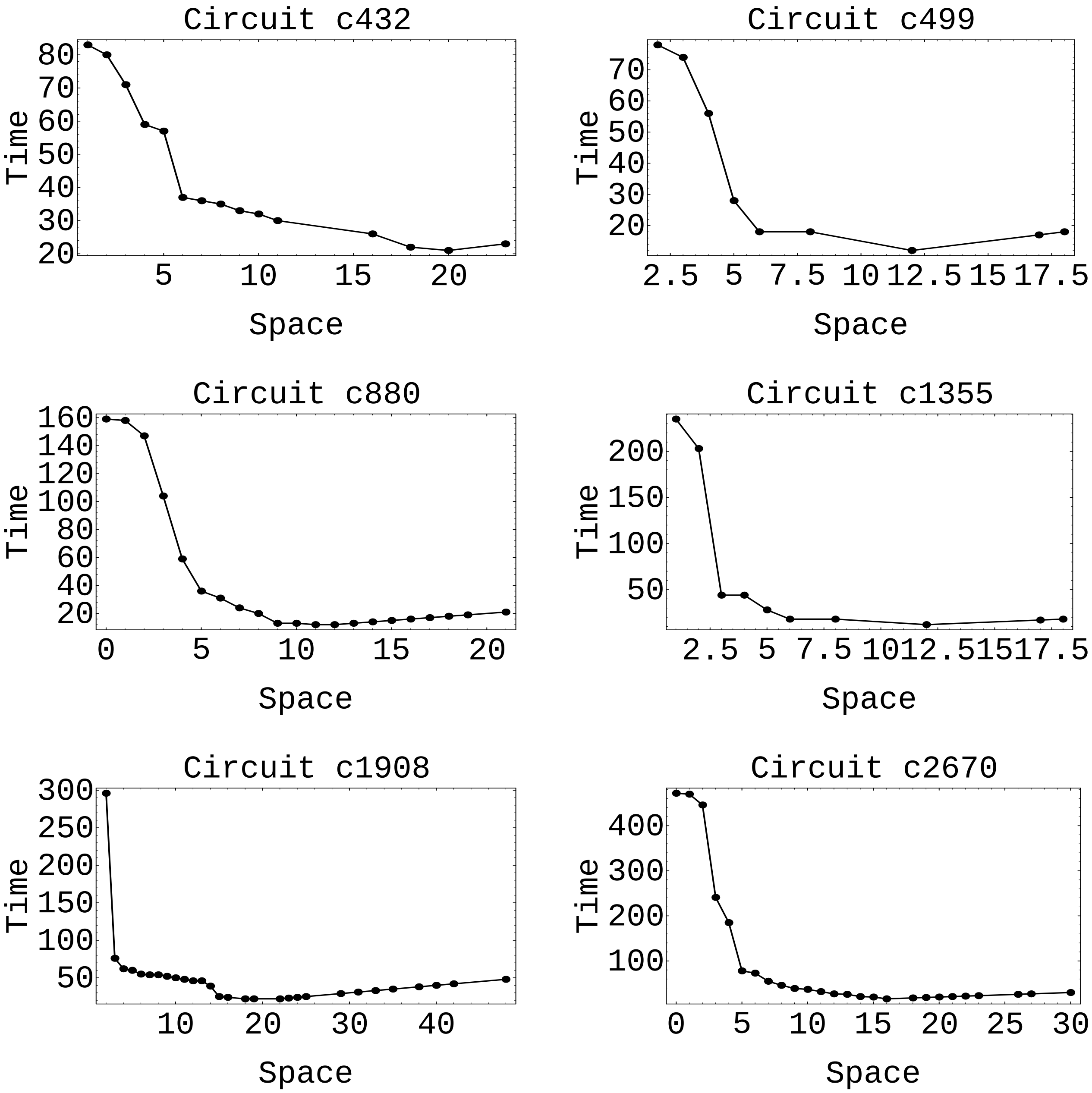

Fig. 13. Time/Space tradeoff for c432 (196 variables), c499 (243 variables), c880 (443 variables), c1355 (587 variables), c1908 (913 variables) and c2670 (1426 variables). Time is measured by the maximum of the separator width and the cutset size and space by the maximum separator width.

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. For instance, the root node for the secondary join tree with separator width 20 (Figure 14) has size 32 while all other nodes have sizes in the range from 2 to

21. When reducing the separator width to 3 the secondary join tree (Figure 16) has the root node with size 172 while all other nodes have sizes in the range from 2 to 4.

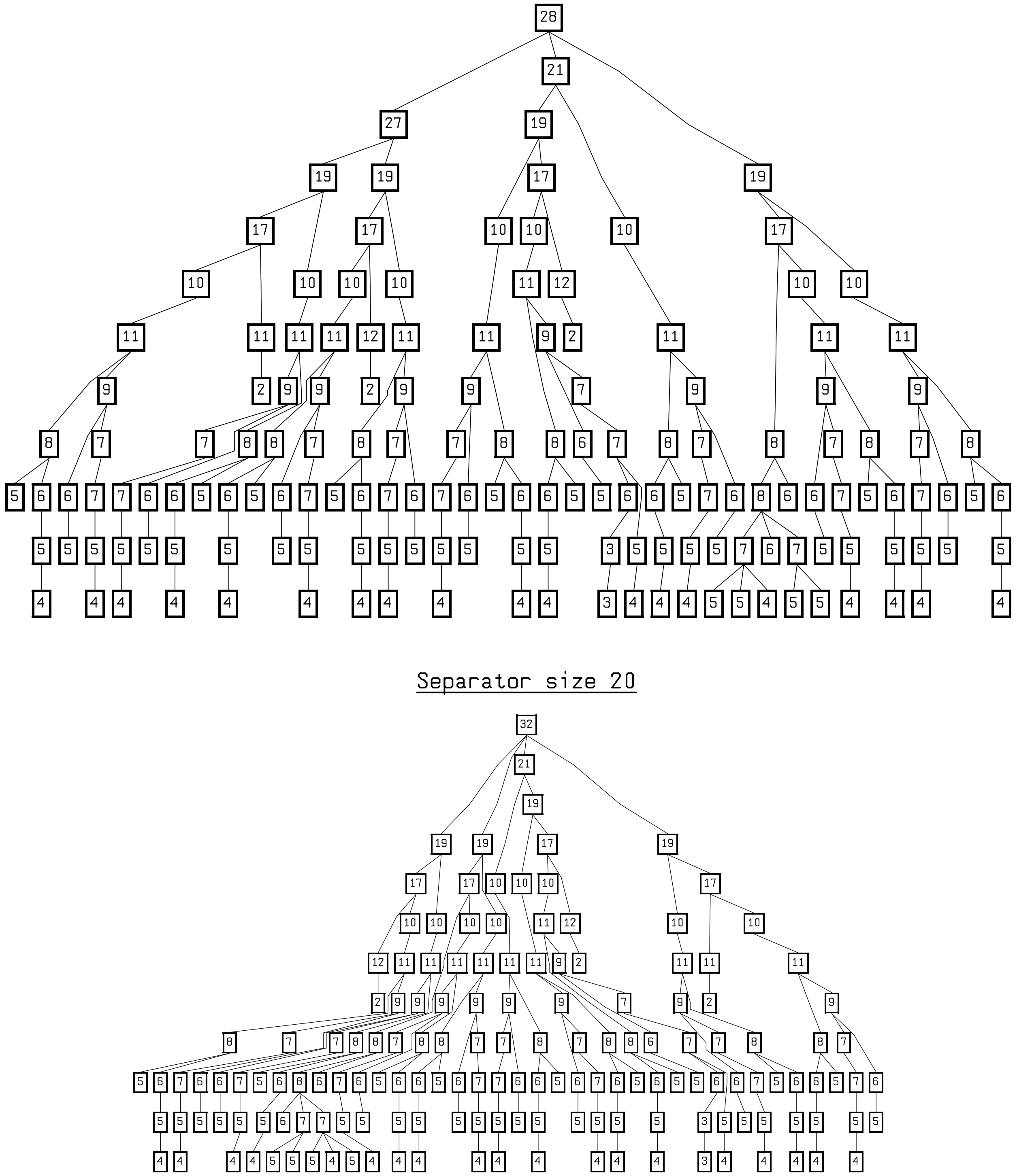

Fig. 14. Secondary trees for c432 with separator widths 23 and 20

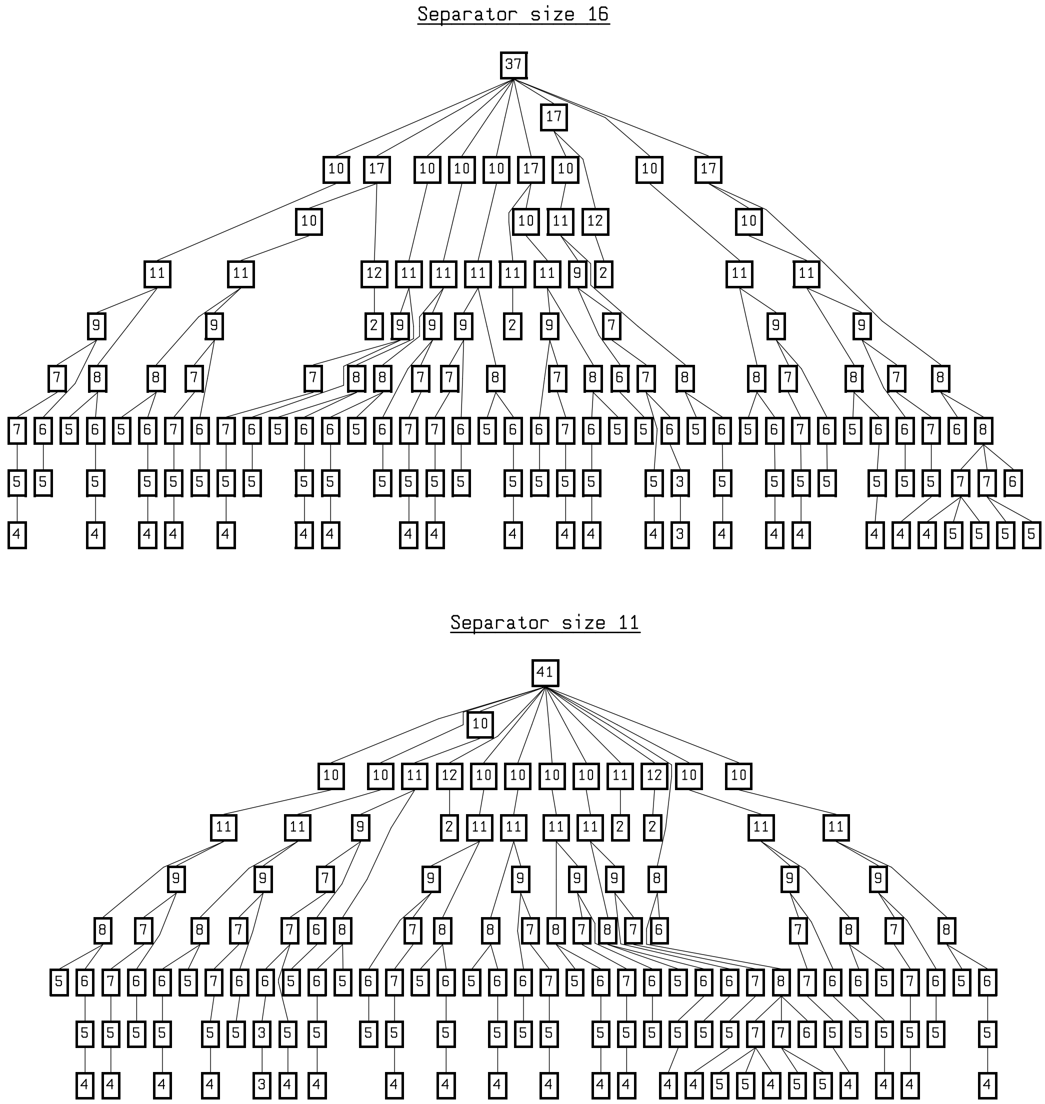

Fig. 15. Secondary trees for c432 with separator widths 16 and 11

7 Related work

The cycle-cutset scheme for probabilistic inference was introduced by Pearl [11] and for constraint networks by Dechter [19]. It was further improved and extended for probabilistic reasoning by [15,25].

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 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-

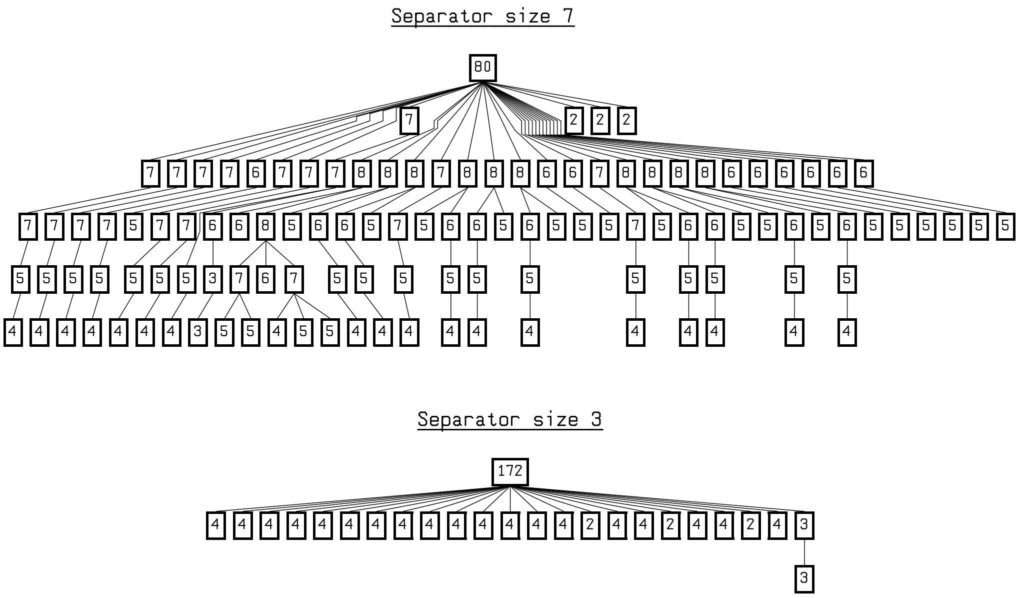

Fig. 16. Secondary trees for c432 with separator widths 7 and 3.

clustering beyond the cycle-cutset scheme appears in the work of Jegou [26] for constraint networks and in the works of [5,27] for probabilistic networks. In the work of Jegou various heuristic are presented, aiming at creating a hybrid algorithm having improved time performance. In [5], 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 [27] the idea is applied to the pathfinder system, where the conditioning set is restricted to the set of diseases.

Finally in [28], a scheme that apply this idea 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 presented in this paper differs in depth and scope. We provide a systematic parameterized hybrid scheme that can be matched to user's application and resources, analyze its time-space tradeoff complexity, show applicability across a wide variety of reasoning frameworks and provide some empirical evidence to its impact. In addition, the particular hybrid idea proposed here is different; rather than applying conditioning globally, we apply conditioning locally within each clique, which may lead to superior time-space guarantees.

8 Summary and Conclusions

Inference algorithms are time and space exponential in the size of the relationships they record 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 (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 the application and the available resources.

Specifically, we have shown that constraint network 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 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. Each clique defines a subproblem and each separator defines the size of the tables necessary for storing the projected solutions to the subproblems. Each subproblem may be solved by any algorithm. In particular, (1) by straightforward search algorithm like backtracking, for which the time complexity is worst-case exponential in the size of the clique, (2) by cycle-cutset conditioning, for which the time complexity is worst-case exponential in the size of the cycle cutset for the subgraphs defined by the cliques.

We address the empirical issue of applicability to real-life of tree-clustering, conditioning, or their hybrids. To that end, we studied the structural parameters of 11 benchmark circuits widely used in the fault diagnosis and testing community [9]. Through those parameters we may predict for each circuit, the limits and potential of (1) pure tree clustering, (2) pure cycle-cutset conditioning and (3) hybrids of tree-clustering and cycle-cutset conditioning. The controlling parameters allow assessing in advance, the complexity of most reasoning tasks on those circuits, and determining the appropriate hybrid level of join-tree clustering and cycle-cutset methods.

We observed that the join-trees of the circuits all shared the unexpected property that the majority of cliques sizes are relatively small while few cliques are distinctly large. 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 we show present worst-case guarantees of the corresponding algorithm. It is still not necessary that the bounds are tight nor that they correlate well with average case performance. This analysis should be extended in the future to include actual implemnetations and testing of the involved algorithms, a major effort outside the scope of this paper.

References

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

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

[3] R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.

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

[5] 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.

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

[7] 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.

[8] 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.

[9] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN, distributed on a tape to participants of the Special Session on ATPG and Fault Simulation, Int. Symposium on Circuits and Systems, June 1985; partially characterized in F. Brglez, P. Pownall, R. Hum, Accelerated ATPG and Fault Grading via Testability Analysis. In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 695–698, June 1985.

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

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

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

[13] 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.

[14] A. Darwiche. Model-based diagnosis using structured system descriptions. *Journal of Artificial Intelligence Research (JAIR)*, pages 165–222, 1998.

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

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

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

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

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

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

[21] F. Bacchus and A Grove. Graphical models for preferences and utility. In *Uncertainty in AI (UAI-95)*, pages 3–10, 1995.

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

[23] R.D. Shachter. Evaluating influence diagrams. *Operations Research*, 34, 1986.

[24] F. Jennsen and F. Jennsen. Optimal junction trees. In *Uncertainty in Artificial Intelligence (UAI-95)*, pages 360–366, 1994.

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

[26] 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.

[27] H. J. Suermondt, G. F. Cooper, 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.

[28] 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.

A Structural parameters of primary trees

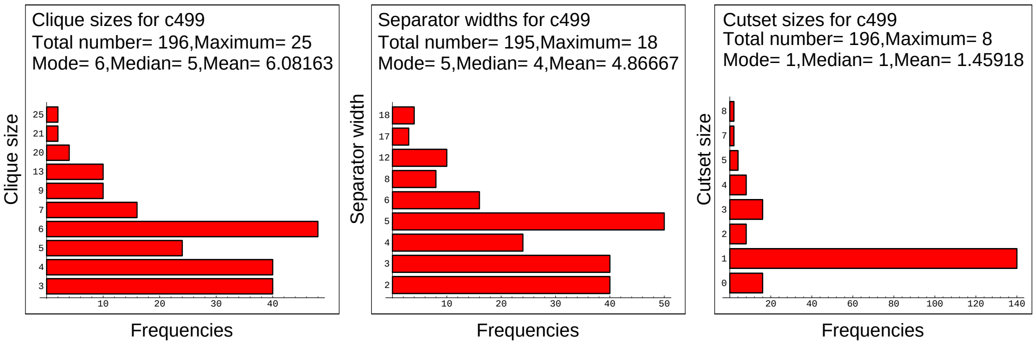

Fig. A.1. Histograms of the cliques sizes the separator widths and the cutsets sizes of the primary join tree for circuit c499 (243 variables).

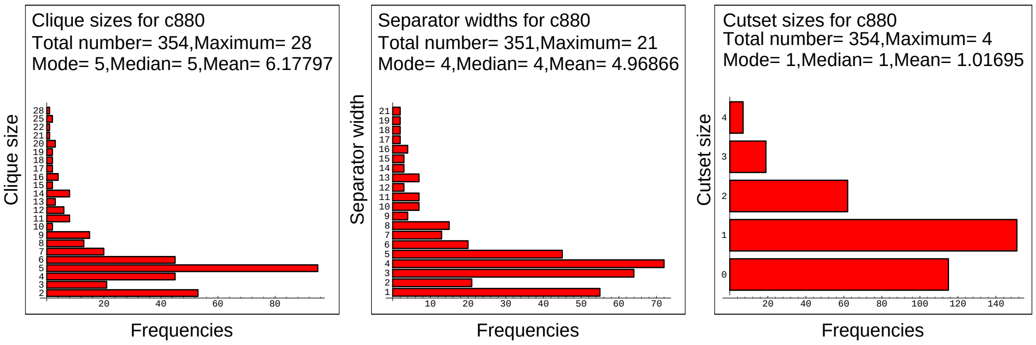

Fig. A.2. Histograms of the cliques sizes the separator widths and the cutsets sizes of the primary join tree for circuit c880 (443 variables).

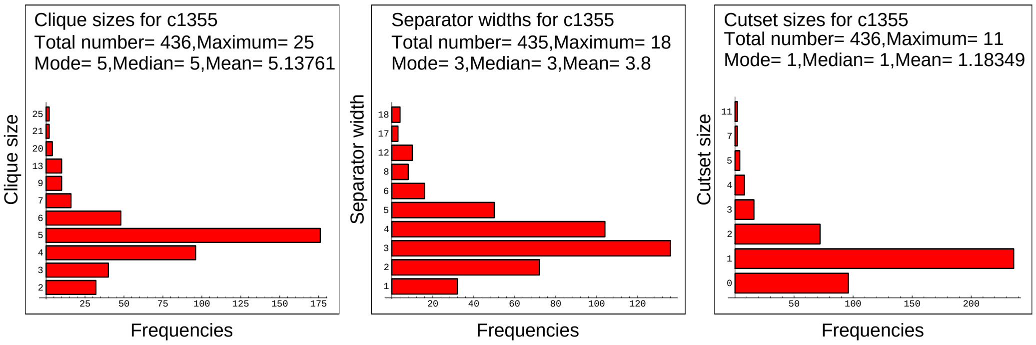

Fig. A.3. Histograms of the cliques sizes the separator widths and the cutsets sizes of the primary join tree for circuit c1355 (587 variables).

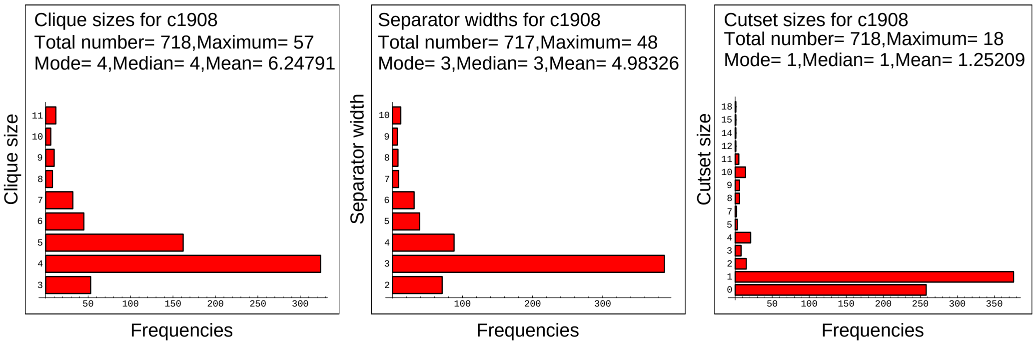

Fig. A.4. Histograms of the cliques sizes (0.9th quantile range), the separator widths (0.9th quantile range) and the cutsets sizes of the primary join tree for circuit c1908 (913 variables).

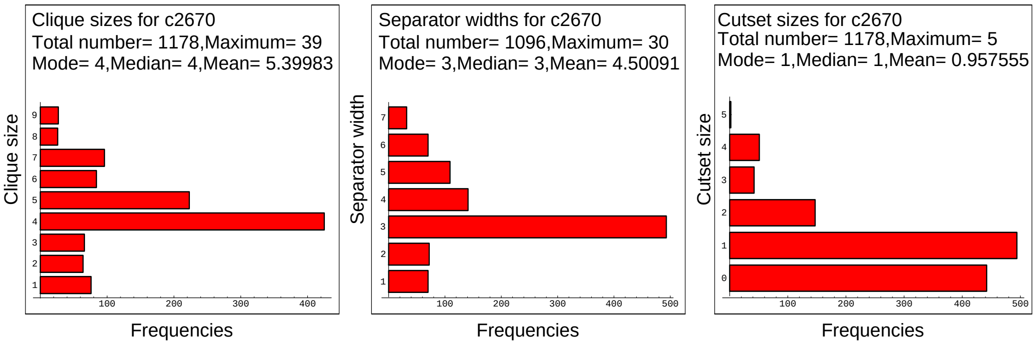

Fig. A.5. Histograms of the cliques sizes (0.9th quantile range), the separator widths (0.9th quantile range) and the cutsets sizes of the primary join tree for circuit c2670 (1426 variables).

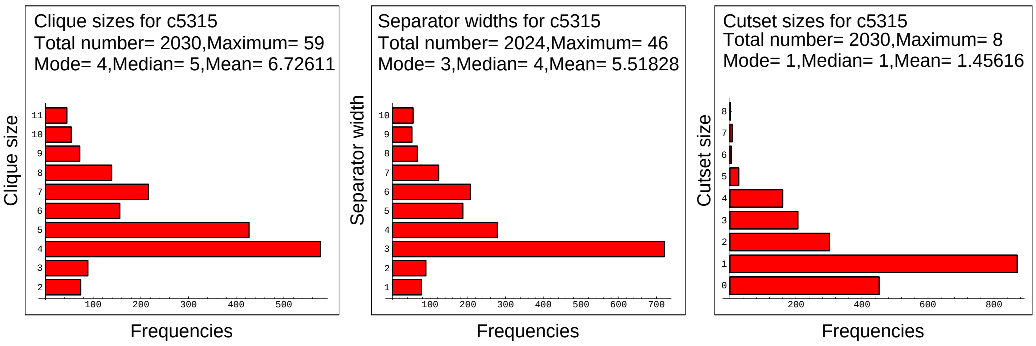

Fig. A.6. Histograms of the cliques sizes (0.9th quantile range), the separator widths (0.9th quantile range) and the cutsets sizes of the primary join tree for circuit c5315 (2485 variables).

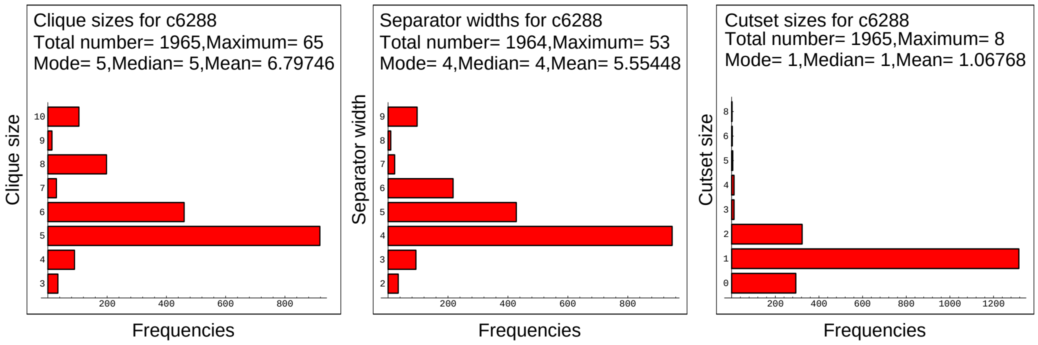

Fig. A.7. Histograms of the cliques sizes (0.9th quantile range), the separator widths (0.9th quantile range) and the cutsets sizes of the primary join tree for circuit c6288 (2448 variables).

Table A.1
Maximum clique sizes (C) and maximum separator widths (S) for the join trees obtained by tree clustering with various ordering heuristics

Circuit	#variables	Causal Ordering		Maximum Cardinality		Minimum Width		Minimum Degree	
		C	S	C	S	C	S	C	S
c17	11	5	3	4	2	4	3	3	2
c432	196	63	62	45	44	51	45	28	23
c499	243	68	66	43	42	33	32	25	18
c880	443	66	64	51	49	66	64	28	21
c1355	587	74	73	47	46	37	36	25	18
c1908	913	139	138	65	63	141	139	57	48
c2670	1426	150	149	82	80	154	152	39	30
c3540	1719	270	269	117	115	294	293	114	89
c5315	2485			172	169			59	46
c6288	2448			277	276			65	53
c7552	3719							58	45

B Ordering Heuristics

As a side effect of our experiments we observed a dramatic difference between the effect of ordering on the resulting primary join tree. In particular the max-cardinality heuristic was shown to be much inferior to the min-degree ordering. Four ordering heuristics were considered: (1) causal ordering,

Table A.2
Number of added edges (N) and the clustering cpu-seconds (T) for various orderings.

Circuit	#variables	Causal Ordering		Maximum Cardinality		Minimum Width		Minimum Degree	
		N	T	N	T	N	T	N	T
c17	11	7	0.15	2	0.017	3	0.05	1	0.033
c432	196	7094	154.967	3425	51.916	1895	29.384	869	22.934
c499	243	7348	137.883	3813	65.617	2289	36.633	944	27.483
c880	443	11286	287.067	6788	153.283	9002	257.4	1678	83.634
c1355	587	15241	465.4	9951	302.683	6695	198.433	1272	104.783
c1908	913	60223	4748.73	24445	1147.2	39746	2559.95	4356	418.367
c2670	1426	115519	17457.4	22413	1350.17	60239	5933.17	4381	967.05
c3540	1719	225016	67385.6	64565	5943.72	148251	33038.9	17043	3650.98
c5315	2485			88714	20595.6			11486	5665.68
c6288	2448			291137	1.4805e+05			14311	5415.73
c7552	3719							14968	20301.3

(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 is exactly like min-width except that we connect neighbors of selected nodes, and causal ordering is just a topological sort of the DAG.

Tree clustering is implemented using each of the four orderings on each of the benchmark circuits of table 1. Table A.1 gives the maximum separator widths and clique sizes for each method on all circuits. We note that among the four methods, the min-degree ordering has by far the best performance as it yields the smallest cliques sizes and separators. The table shows the maximum separator widths to be tightly correlated with the maximum clique sizes. As an example, for circuit c3540, which has 1719 variables, the separator width for the minimum degree method is 89. Causal-ordering leads to a separator width that is 300% greater than the maximum degree, the maxium-cardinality 130%, and minimum-width 330%. Unlike the three other methods, the minimum degree leads to separator width that grows only slowly with the size of the circut. Indeed, the minimum degree was the only method that could scale-up to the largest size circuit. For circuit c6288 (2448 variables) the separator width for minimum-degree is only 53 while with maximum cardinality is 276 (520% larger than minimum degree).

Table A.2 gives the number of added edges and the cpu-seconds for clustering with each ordering on all circuits. The number of added edges serves as an implementation-independent basis for comparing the efficiency of the various clustering methods. The cpu-seconds are tightly correlated with the number of added edges.

C Structural parameters of secondary trees (Additional data)

Table C.1
Structural parameters of primary and secondary trees for c432 (196 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
23	157	28	17	6.22436	7.43312	1.92357
20	156	32	21	6.11613	7.33333	1.85897
18	155	33	22	6.02597	7.25161	1.8129
16	151	37	26	5.70667	6.96689	1.6755
11	147	41	30	5.42466	6.72109	1.54422
10	145	43	32	5.34722	6.66207	1.45517
9	144	44	33	5.31469	6.63889	1.41667
8	126	71	35	4.784	6.30159	1.56349
7	117	80	36	4.53448	6.17094	1.62393
6	106	91	37	4.27619	6.08491	1.71698
5	86	111	57	3.87059	6.10465	1.75581
4	65	132	59	3.5	6.46154	1.93846
3	25	172	71	2.66667	10.4	3.64
2	6	191	80	1.4	33.8333	13.5
1	4	193	83	1.0	49.75	20.75

Table C.2
Structural parameters of primary and secondary trees for c499 (243 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
18	196	25	8	4.86667	6.08163	1.45918
17	192	29	8	4.59162	5.83333	1.47917
12	189	49	11	4.39362	5.65608	1.4127
8	179	59	18	3.96629	5.30168	1.45251
6	171	67	18	3.77647	5.17544	1.52047
5	155	83	28	3.54545	5.09032	1.38065
4	105	139	56	2.84615	5.13333	1.52381
3	81	163	74	2.5	5.46914	1.90123
2	41	203	78	2.0	7.87805	2.87805

Table C.3
Structural parameters of primary and secondary trees for c880 (443 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
21	354	28	4	4.94051	6.17797	1.01695
19	352	32	4	4.849	6.09375	1.01136
18	350	34	4	4.76791	6.02	1.0
17	348	43	5	4.69164	5.95115	1.00575
16	346	45	6	4.62029	5.88728	1.0
15	342	58	9	4.4868	5.76901	0.991228
14	339	62	10	4.39349	5.68732	0.99115
13	336	66	11	4.30746	5.6131	0.994048
12	329	71	11	4.12195	5.45593	0.993921
11	326	75	12	4.04923	5.39571	0.993865
10	319	82	13	3.89623	5.27273	1.0
9	312	88	13	3.75884	5.16667	1.00321
8	308	103	20	3.69055	5.11688	1.01623
7	293	118	24	3.46918	4.96928	1.03072
6	280	131	31	3.30466	4.875	1.06786
5	260	151	36	3.09653	4.78846	1.08846
4	215	196	59	2.69626	4.74419	1.16279
3	143	268	104	2.03521	5.11888	1.4965
2	79	349	147	1.24359	6.83544	2.20253
1	58	380	158	0.964912	8.58621	2.7931
0	3	429	159	0.0	147.667	53.6667

Table C.4
Structural parameters of primary and secondary trees for c1355 (587 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
18	436	25	11	3.8	5.13761	1.18349
17	432	29	11	3.66821	5.01852	1.18981
12	429	49	11	3.57477	4.93473	1.14452
8	419	59	18	3.37321	4.76611	1.15513
6	411	67	18	3.28293	4.70316	1.17762
5	395	83	28	3.17259	4.65063	1.10886
4	345	139	44	2.90698	4.6	1.14783
3	241	163	44	2.43333	4.85892	1.64315
2	105	419	203	1.69231	7.26667	2.92381
1	33	555	235	1.0	18.7576	7.12121

Table C.5
Structural parameters of primary and secondary trees for c1908 (913 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
48	718	57	18	4.98326	6.24791	1.25209
42	717	65	18	4.92318	6.18968	1.2371
40	716	68	18	4.87133	6.13966	1.22765
38	715	69	18	4.82213	6.09231	1.21958
35	712	74	18	4.68214	5.95787	1.18399
33	711	76	19	4.63944	5.91702	1.16596
31	710	77	20	4.59944	5.87887	1.1493
29	708	82	20	4.52475	5.80791	1.13418
25	707	83	20	4.49008	5.77511	1.12447
24	706	84	20	4.46099	5.74788	1.12323
23	704	88	22	4.40541	5.69602	1.11222
22	703	90	22	4.37892	5.67141	1.10953
19	701	92	22	4.32857	5.62482	1.10271
18	700	94	22	4.30758	5.60571	1.09857
16	696	100	24	4.22878	5.53448	1.08333
15	692	104	25	4.16064	5.47399	1.05925
14	678	118	39	3.93648	5.27729	0.914454
13	666	130	46	3.75489	5.12012	0.795796
12	660	138	46	3.67071	5.04848	0.777273
11	656	143	48	3.61985	5.0061	0.778963
10	654	145	50	3.59724	4.98777	0.776758
9	642	158	52	3.47738	4.89408	0.758567
8	635	166	54	3.4164	4.84882	0.76378
7	627	174	54	3.35783	4.80861	0.767145
6	618	185	55	3.3047	4.7767	0.775081
5	587	217	60	3.16212	4.7121	0.749574
4	548	258	62	3.03108	4.69161	0.775547
3	460	350	76	2.84532	4.82391	0.897826
2	72	788	296	2.0	14.6528	4.95833

Table C.6
Structural parameters of primary and secondary trees for c2670 (1426 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
30	1178	39	5	4.19116	5.39983	0.957555
27	1177	43	5	4.16922	5.37893	0.958369
26	1176	48	5	4.14979	5.36054	0.958333
23	1175	49	5	4.13118	5.34298	0.959149
22	1171	49	5	4.06667	5.28266	0.962425
21	1168	49	5	4.02057	5.23973	0.961473
20	1161	74	5	3.9181	5.1447	0.966408
19	1156	86	6	3.84848	5.08045	0.966263
18	1155	87	7	3.83536	5.0684	0.966234
16	1151	106	14	3.78609	5.02346	0.97046
15	1142	121	20	3.68975	4.93695	0.971979
14	1138	126	21	3.64996	4.90158	0.971002
13	1130	134	26	3.57662	4.83717	0.976106
12	1128	136	27	3.55989	4.8227	0.97695
11	1118	147	32	3.48433	4.7585	0.973166
10	1110	156	37	3.43012	4.71351	0.976577
9	1101	168	39	3.37636	4.6703	0.97911
8	1089	180	46	3.31434	4.62259	0.972452
7	1069	201	55	3.22659	4.5594	0.981291
6	1037	241	73	3.11004	4.48409	1.0
5	967	264	78	2.90062	4.37435	0.951396
4	858	445	185	2.63361	4.29487	0.952214
3	717	569	241	2.36453	4.35286	1.05439
2	224	1129	446	0.959641	7.33036	2.36607
1	152	1207	470	0.463576	9.85526	3.34211
0	82	1251	472	0.0	17.4146	6.15854

Table C.7
Structural parameters of primary and secondary trees for c6288 (2448 variables) ; Max. separator width (Smx), #cliques (N), max clique size (Cmx), max cutset size (CSmx), average separator width (Sav), average clique size (Cav), average cutset size (CSav).

Smx	N	Cmx	CSmx	Sav	Cav	CSav
53	1965	65	8	5.55448	6.79746	1.06768
30	1952	162	19	5.33265	6.58402	1.04816
16	1909	285	19	4.96226	6.24201	1.02252