Methods for Constructing Balanced Elimination Trees and Other Recursive Decompositions

Kevin Grant and **Michael C. Horsch**
Dept. of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
Email: kjg658@mail.usask.ca and horsch@cs.usask.ca

Abstract

A conditioning graph is a form of recursive factorization which minimizes the memory requirements and simplifies the implementation of inference in Bayesian networks. The time complexity for inference in conditioning graphs has been shown to be **O**($n \exp(d)$), where d is the depth of the underlying elimination tree. We demonstrate in this paper techniques for building small elimination trees. We give a simple method for deriving elimination trees for Darwiche et al.'s dtrees. We also give two new heuristics for building small elimination trees. We show that these heuristics, combined with a constructive process for building e-trees produces the smaller trees.

Introduction

When programmers wish to use Bayesian networks in their applications, the standard convention is to include the entire network, as well as an inference engine to compute posteriors from the model. Algorithms based on junction-tree message passing (Lauritzen & Spiegelhalter 1988) or variable elimination (Zhang & Poole 1994; Dechter 1999) have a high space requirement and are difficult to code. Furthermore, application programmers not wishing to implement an inference method must import large general-purpose libraries.

Compiled versions of Bayesian networks overcome this difficulty to some extent. Query-DAGs (Darwiche & Provan 1996) precompute probability equations that are parameterized by evidence variables, and stores them as DAGs. New evidence changes the parameter values and the equations. The inference engine for these systems is very lightweight, reducing system overhead substantially. And the interface to the system is sufficiently easy - the user can either set evidence nodes or query probabilitistic output nodes. However, although the abstraction provided by Q-DAGs makes them universally implementable, their size may be exponential in the size of the network.

Recently, we proposed *conditioning graphs* (Grant & Horsch 2005). Conditioning graphs combine the linear space requirements of conditioning with the simplicity of Q-DAGs. Its components consist of simple node pointers and floating point values; no high-level elements of Bayesian network computation are included. As well, the inference algorithm for conditioning graphs is a small recursive algorithm, easily implementable on any architecture.

The time complexity of inference using a conditioning graph is exponential on the height of its underlying elimination tree. Hence, minimizing the height of these elimination trees is of particular interest. Producing balanced recursive decompositions has been previously investigated by Darwiche et al. (Darwiche 2000; Darwiche & Hopkins 2001). These documents provide two methods for balancing dtrees (a recursive decomposition where the size of the cutset at each node is unrestricted, but every internal node must have two children). The first method involves constructing an unbalanced dtree, and subsequently balancing it using contraction methods. The second involves hypergraph partitioning, which combines construction and balancing into a single step.

We demonstrate a simple transformation between dtrees and elimination trees, and show that the time complexity of inference in the elimination trees after conversion is the same as in the dtree. Thus, any algorithm for building well-balanced dtrees also contributes well-balanced elimination trees. We also suggest two new heuristics for constructing recursive decompositions, based on greedy search, and show empirically that these are better than the method suggested by (Darwiche & Hopkins 2001) for tested networks when no caching is employed.

The rest of the document is structured as follows. We begin with a review of elimination trees and their construction, which is followed by a discussion of the transformation from a dtree to a elimination tree (e-tree). We then introduce two heuristics for building e-trees, and show that e-trees constructed using these heuristics are superior e-trees constructed from balanced dtrees. We close with a summary and future work.

Background

A *Bayesian network* is a tuple $\langle \mathcal{G}, \boldsymbol{\phi} \rangle$, where $\mathcal{G} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle$ is a directed acyclic graph over random variables $\boldsymbol{V} = \{V_1, ..., V_n\}$. $\boldsymbol{\phi} = \{\phi_1, ..., \phi_n\}$ is a set of potentials, where ϕ_i is a conditional probability distribution of V_i given its parents in $\mathcal{G}$.

An *elimination tree* (Grant & Horsch 2005) or *e-tree* over

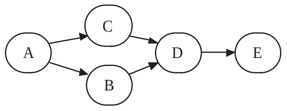

Figure 1: An example Bayesian network.

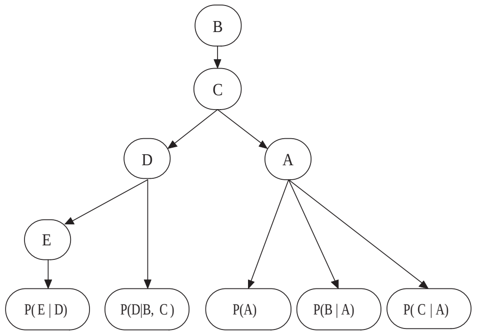

Figure 2: The network from Figure 1, arranged in an elimination tree.

a Bayesian network is a tree where each leaf corresponds to a CPT in the network, and each non-leaf corresponds to a random variable from the network. The tree is structured such that for any node N in the tree, the variable at N and N's ancestors d-separate all variables from one child of N from all variables in another child of N. Figure 2 shows an e-tree for the network shown in Figure 1.

Elimination trees are a recursive factorization of Bayesian networks. Other recursive factorizations exist, such as recursive decomposition (Monti & Cooper 1996) and recursive conditioning (Darwiche 2000). The primary difference is that these factorizations restrict the number of children at each internal node to two, while no restriction is made on the number of variables at each internal node. In contrast, elimination trees restrict the number of variables at each internal node to one, but have no restriction on the number of children.

To compute a probability from an elimination tree, we traverse it depth-first, generating a context as we descend, such that when we reach a leaf node, we can extract a probability from the corresponding CPT. Values returned from the children of a node are multiplied. If the variable at a node is not observed, then we condition over that variable. When the variable at a node is an evidence variable, its value is fixed, which affects all calculations in its descendants.

A *conditioning graph* is an e-tree with additional arcs connecting variable nodes to leaf nodes. The basic algorithm for computing probabilities is the same, but the additional structure simplifies the basic algorithm to increase its portability.. Space restrictions preclude a more detailed explanation of this approach, but see (Grant & Horsch 2005) for more details. The time complexity for inference in an e-tree is exponential on the height of the tree.

Elimination trees have a close correspondence with elimination algorithms (Zhang & Poole 1994; Dechter 1999). The algorithm for building an elimination tree parallels variable elimination, where an internal node represents the marginalization of its variable label, and the children of the node represent the distributions that would be multiplied together. Thus, an internal node is labeled with a variable, but represents a distribution. Figure 3 gives a simple algorithm for constructing an elimination tree from a Bayesian network $\langle \mathcal{G}, \mathbf{\Phi} \rangle$. In the algorithm, we use $dom(T)$ to represent the union of all CPT domains from the leaves of T's subtree.

Notice that the algorithm in Figure 3 returns a set of trees, rather than a single tree. In the event that the network is not connected, the number of disconnected components will correspond to the number of trees returned by *elimtree*. We assume that the elimination tree is connected (that is, the algorithm returns a single tree).

```
elimtree($\langle \mathcal{G} = \langle \boldsymbol{V}, \boldsymbol{E} \rangle, \mathbf{\Phi} \rangle$)
    $\boldsymbol{T} \leftarrow \{\}$
    for each $\phi \in \mathbf{\Phi}$ do
      Construct a leaf node $T_\phi$ containing $\phi$
      Add $T_\phi$ to $\boldsymbol{T}$
    for each $V_i \in \boldsymbol{V}$ do
      Select the set $\boldsymbol{T}_i = \{T \in \boldsymbol{T} | V_i \in dom(T)\}$
      Remove $\boldsymbol{T}_i$ from $\boldsymbol{T}$
      Construct a new internal node $t_i$ whose children are $\boldsymbol{T}_i$
      Label $t_i$ with $V_i$, and add it to $\boldsymbol{T}$
    return $\boldsymbol{T}$
```

Figure 3: The code for generating an elimination tree from a Bayesian network.

Given an e-tree of height d, the time complexity of computing a probability from the tree is **O**$(n \exp(d))$, where $d = n$ in the worst case. Although the worst case rarely occurs, it demonstrates the importance of an e-tree with good structure. The dtree structure due to Darwiche et al. is related to our e-tree structure, and methods for constructing well-balanced dtrees have been proposed (Darwiche 2000; Darwiche & Hopkins 2001). The following section demonstrates a relationship between dtrees and e-trees, such that we can take advantage of these algorithms for building balanced dtrees when constructing e-trees. Also, the above algorithm for constructing e-trees suggests that the complexity of the e-tree is a function of the variable ordering. In a subsequent section, we examine how to construct good variable orderings such that e-trees can be computed directly, without resorting to secondary balancing methods (these are described in the next section).

Dtrees to Elimination Trees

As mentioned, a *dtree* is a recursive decomposition of a Bayesian network, where each internal node has two children. The number of variables at each node is not restricted to one variable as it is in e-tree nodes. Figure 4 shows a possible dtree for the network of Figure 1.

The time complexity of computing probabilities in a dtree (when no caching is used) is **O**$(n \exp(wd))$ where w is the size of the largest cutset (set of variables at a node), and

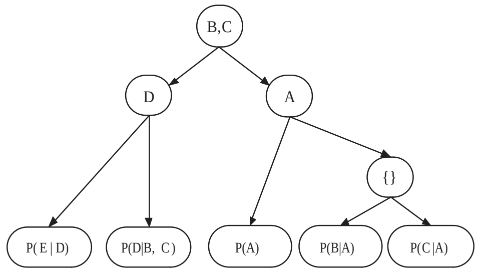

Figure 4: A dtree for the Bayesian network in Figure 1

d is the maximum depth of the tree (the quantity measured is the number of recursive calls made). If the tree is balanced, then $d = \log n$. There are two well-established algorithms for balancing dtrees. The first (Darwiche 2000) involves constructing a dtree using variable elimination (in the same manner as elimination trees are constructed), and then balancing them using contraction (Miller & Reif 1985). The second involves directly computing balanced trees using hypergraph partitioning (Darwiche & Hopkins 2001).

The similarity of dtrees and e-trees suggests that a well-balanced dtree might lead to a well-balanced e-tree. Transforming a dtree to an e-tree is straightforward. We show a transformation method, and then show that the complexity of the resulting e-tree is the same as the original dtree.

There are several things to note about the conversion. First, the leaf nodes of a dtree and an e-tree are identical, that is, they correspond to a CPT from the Bayesian network. Second, the cutsets of a dtree do not include leaf variables from the original Bayesian network. A cutset in a dtree represents a subset of the intersection of variables from the leaf CPTs of its subtree. Hence, since a leaf variable in a Bayesian network is only defined in one CPT, it follows that it never appears in a cutset.

The conversion process is as follows:

1. For each leaf variable in the original Bayesian network, create a new node containing that variable, and insert it as the parent of the node containing that variable's CPT.
2. If a node has no variables in its cutset, then the children of this node become children of the parent node.
3. If a node has k variables in its cutset, where $k > 1$, then it creates a chain of $k - 1$ nodes beneath it, and assigns each one a variable from the cutset.

This algorithm converts the dtree of Figure 4 to the e-tree shown in Figure 2. We now prove that the time complexity of inference over the two structures is the same.

Lemma 1. *After converting a dtree of depth d and cutset width w, the resulting e-tree has a depth of d_2, where $d_2 \leq d * w + 1$.*

Proof Sketch: Follows from the node transformations. Adding a leaf variable above a leaf node increases any path in the tree by at most 1. Absorbing nodes does not increase the height. Creating a chain out of a set of cutset variables increases the length of a path from 1 to w (since the cutset is at most size w). Hence, since the the number of nodes in any path is at most d, the maximum length of a path in the e-tree is $d * w + 1$.

Theorem 1. *The time complexity to compute posterior probabilities using a dtree is the same as the time complexity using an e-tree constructed from that dtree.*

Proof. The time complexity of inference using a dtree of height d and width w is **O**$(n\ exp(wd))$ (Darwiche 2000). From the lemma, the e-tree constructed from such a dtree is of height $d * w + 1$. Since the time complexity of inference over an e-tree is exponential on its height, it follows that the two structures share the same complexity. $\square$

We generate elimination trees from balanced dtrees using both techniques discussed by Darwiche et al. (Darwiche & Hopkins 2001): generate a tree, then balance them using contraction; and generate a balanced dtree using hypergraph partitioning. We compare these e-trees to those constructed using the methods described in the next section. The results of this comparison are given in the *Evaluation* section.

In closing this section, we note that the transformation from a dtree to an e-tree can be reversed. This is important, since the complexity of computing over a dtree is a function of two factors: the height of the tree, and the width of the cutsets. In Darwiche et al. (Darwiche & Hopkins 2001), the authors explicitly compare the construction algorithms by each term, but not by product of these two factors. It is the product of these terms that determines the time complexity of computing over the structure in the absence of any caching. In contrast, the complexity of computing over e-trees is a function only of height. Therefore, by minimizing the complexity of an e-tree, we are minimizing the aforementioned product in a dtree. Therefore any method developed to build good e-trees can be used to build good dtrees. This is especially important if the dtree will be used without any caching of intermediate results.

Better Elimination Orderings

In inference algorithms based on junction trees or variable elimination, a good elimination ordering results in small cliques, or small intermediate distributions. Finding an optimal elimination ordering is NP-hard; heuristic methods, which are relatively fast, have been shown to give good results in most cases. Starting from a moralized graph, the *min-fill* heuristic chooses to eliminate the variable which would require the fewest edges to be added to the network during triangulation; the *min-size* heuristic chooses to eliminate the variable which would minimize the number of neighbouring variables (Kjaerulff 1990; Huang & Darwiche 1996).

These heuristics are not necessarily well suited for recursive decomposition techniques, especially if intermediate results are not cached (Darwiche 2000). They try to minimize clique size, which is not directly related to the time complexity of inference over a decomposition structure such as an e-tree. Consider the example shown in Figure 5. Using the *min-fill* heuristic, we will always remove a node from the end of the chain, which leads to the possibility of an elimination ordering such as G, F, E, D, C, B, A. This ordering

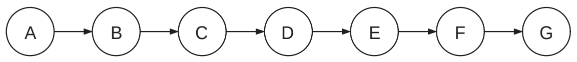

Figure 5: For this Bayesian network, an elimination ordering that is optimal for inference based on junction trees is the worst case for methods based on decomposition structures.

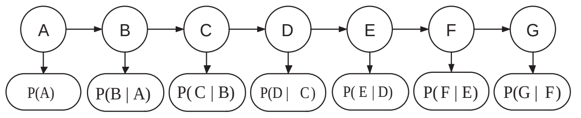

Figure 6: A worst case e-tree for the Bayesian network in Figure 5, constructed using the min-fill heuristic.

is optimal for inference methods based on junction trees or variable elimination. However, it is the worst case for inference over e-trees and dtrees. Figure 6 shows the elimination tree generated from this elimination ordering (the corresponding dtree is exactly the same, minus the node containing the leaf variable). The height of the e-tree corresponds to the number of nodes in the network, making the complexity of inference $\mathbf{O}(n \exp(n))$.[1]

Darwiche shows that a bad ordering can be repaired using rake and compress methods (Miller & Reif 1985). However, we take a more direct approach, trying to measure (and minimize) the height of the resulting e-tree, which directly affects the time complexity of inference over it. Recall that an e-tree is constructed iteratively, and when a variable is chosen as the root of a new e-tree, all partial e-trees that include this variable in their definition are made children of the chosen variable. We wish to choose the variable such that the resulting e-tree has the smallest height.

The height of an e-tree is determined by taking the current height, and estimating the additional height that would result in later iterations. This is very similar to the way heuristics are used in A* search: the best choice minimizes f, which is the sum of current cost g with estimated remaining cost h. We define $g(T)$ as the current height of a given e-tree T. The estimate $h(T)$ is the number of variables in the domain of T that have not yet been eliminated. This value corresponds exactly to the *min-size* heuristic of classical elimination order generation, and provides a lower bound on the remaining height of the tree.

We define the heuristic function f, as a weighted sum of g and h, so that their effect in the search can be manipulated: $f = (1 - \alpha)g + \alpha h$, where $\alpha \in [0, 1]$. Using $\alpha = 1$ corresponds to using the *min-size* heuristic. Using $\alpha = 0$ corresponds to a heuristic based only on the estimate of the height of the tree using the remaining variables. Excluding the α values from the equation provides a tight lower bound on the eventual height of the resulting tree.

The approach we take in this paper is based on hill-climbing search, using a heuristic that has some similarity to the way heuristics are used in A* search. Because of the similarity, it is important to note that no backtracking occurs, and the resulting ordering is not guaranteed to be optimal. Converting the greedy search to a best-first approach is a simple extension, which we have not fully explored.

The choice of current tree height (g) as a component in our heuristic is an obvious one. However, using the number of remaining variables in the e-tree is only one choice for a lookahead value. Indeed, since it corresponds exactly to the *min-size* heuristic, a natural question to ask is if we can use the *min-fill* heuristic, which is typically preferred in classical variable-ordering applications over *min-size*.

The problem with *min-fill* is that counts edges, rather than variables, so an additive combination of g (which counts height in terms of a number of variables), and *min-fill* would not give a consistent estimate of total height. Furthermore, no simple setting of α can account for the difference in these measures. We resolve this problem by noting that if the number of remaining nodes to be marginalized is n, then the maximum number of necessary fill edges is $f = n(n-1)/2$. Solving for n gives $n = (1 + \sqrt{1 + 8f})/2$. This value derived from *min-fill* can be used as our lookahead value in the heuristic function f.

Finally, when selecting a node, it is very often the case that many variables have the same best f value, especially early in the search. Using the traditional methods, Darwiche and Huang recommend using the *min-fill* algorithm, breaking any ties with the *min-size* algorithm (Huang & Darwiche 1996). However, when working on this project, we found that even with tie-breaking procedures in place, there were still a large number of unresolved ties that had to be broken arbitrarily. To address this issue, we break these ties by choosing one of the best variables at random.

Evaluation

We compare the quality of the e-trees produced by our heuristics to those produced from balanced dtrees. We use both heuristics from the previous section: the first uses h as the *min-size* heuristic, and the second uses the modified *min-fill* value. This comparison is made using several well-known Bayesian networks from the Bayesian network repository.[2] We also follow the approach taken in (Darwiche & Hopkins 2001), and compare our algorithms over several ISAC '85 benchmark circuits.

Because all the heuristics employ random ties breaking, we show results as the mean of 50 trials for each configuration. The bold entries in the tables of results indicate where the mean height of the final tree using our heuristics is superior to the best result from any of the other methods.

Table 1 shows the results of the first comparison, using the benchmark Bayesian networks. In the first column, we show the mean height of e-trees derived from a dtree constructed using the *min-fill* heuristic (Darwiche 2000), without balancing. The second column shows the height of e-trees derived from balanced dtrees, using *contract* (Darwiche 2000). The third column shows the mean height of the e-trees converted

[1]The best possible ordering chosen by min-fill for a chain of variables leads to an e-tree of height $n/2$, which is still linear in the number of variables.

[2]http://www.cs.huji.ac.il/labs/compbio/Repository/.

	DTree Conversion			Best-first search (values indicate α)										
	mf	*mf-bal*	*hmetis*	*0.0*	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1.0*
Barley	21.8	19.52	15.09	19.96	**14.1**	**13.8**	**13.8**	**14.32**	**14.08**	**14.18**	**14.98**	**14.98**	15.24	19.16
Diabetes	60.36	23.28	18.95	50.98	**18.32**	**18.16**	**18.18**	**18.14**	19.48	21.14	20.98	24.1	30.46	52.44
Link	45.74	43.4	48.2	147.8	**40.38**	**39.32**	**39.04**	**39.54**	**40.36**	**40.56**	**38.98**	**39.4**	**39.18**	47.08
Mildew	13.66	12.2	11.02	15.18	**9.42**	**9.34**	**9.28**	**10.34**	**10.44**	**10.3**	**10.72**	**10.56**	**10.64**	12.7
Munin1	23.26	22.16	26.39	42.0	**19.02**	**18.64**	**18.68**	**18.84**	**19.64**	**19.68**	**19.48**	**22.0**	**20.8**	23.46
Munin2	31.44	23.76	26.16	78.4	**15.92**	**15.78**	**15.72**	**16.04**	**16.68**	**17.06**	**18.72**	**20.4**	**21.58**	29.22
Munin3	26.82	21.02	24.48	78.66	**16.0**	**16.26**	**16.78**	**17.4**	**17.62**	**17.9**	**19.0**	**19.06**	**20.46**	27.6
Munin4	27.38	22.3	28.78	90.06	**17.4**	**17.36**	**17.16**	**18.08**	**18.42**	**18.68**	**20.32**	**21.18**	22.9	28.44
Pigs	26.06	24.6	24.23	48.42	**20.72**	**20.56**	**20.58**	**20.16**	**20.62**	**21.32**	**21.26**	**21.42**	**21.8**	25.78
Water	15.82	15.82	16.0	19.92	16.22	**15.1**	**15.0**	**15.0**	**15.36**	16.0	16.0	16.0	16.0	16.8

Table 1: Heights of constructed e-trees on repository Bayesian networks using the modified *min-size* heuristic for lookahead.

	DTree Conversion			Best-first search (values indicate α)										
	mf	*mf-bal*	*hmetis*	*0.0*	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1.0*
c432	41.72	41.4	47.2	73.68	42.3	**40.4**	**38.7**	**38.14**	**38.06**	**39.0**	**39.0**	**39.0**	**39.0**	42.3
c499	47.92	42.12	40.67	77.82	**40.6**	**39.06**	**39.06**	41.08	41.76	45.0	41.0	40.96	40.96	46.22
c880	55.9	51.92	54.35	124.62	**50.62**	**48.3**	**48.42**	**47.58**	**48.12**	**47.52**	**48.04**	**48.22**	**48.42**	56.02
c1355	50.38	50.38	45.78	123.9	**45.18**	**43.4**	**43.08**	**45.08**	**44.88**	**45.0**	**39.0**	**43.0**	**43.0**	51.52
c1908	74.12	72.48	84.8	195.92	87.76	83.06	80.54	76.52	76.0	74.32	73.96	73.76	74.48	78.62

Table 2: Heights of constructed e-trees on ISAC '85 benchmark circuits, also using *min-size* for lookahead.

from a dtree constructed using hypergraph partitioning (Darwiche & Hopkins 2001). The subsequent columns are generated using our modified *min-size* heuristic described above, for varying α values.

From this table, we can make a few observations. Considering only the dtree numbers, it can be observed that it is better to build an e-tree from a balanced dtree, rather than an unbalanced one. Second, our results show that for constructing e-trees (where we are less concerned with the width of the ordering, and more concerned with the height of the e-tree), hypergraph partitioning did not show an overall advantage over a balanced dtree constructed using *min-fill*; the best results were obtained when the hypergraph partitioning software, *hmetis*, was restricted using a parameter that tries to enforce a very strict fair balancing of the nodes between partitions. However, the hypergraph partitioning algorithm was shown by Darwiche and Hopkins to be a better algorithm for constructing good dtrees (Darwiche & Hopkins 2001).

Most notably, our modified *min-size* heuristic consistently outperformed the dtree based constructions, for α values between 0.2 and 0.5. The reductions in e-tree height were between 1 and 8 variables for the networks tested. Considering that the complexity is exponential on the height of the tree, such a reduction is very significant. The best results appear for $\alpha \in [0.2, 0.5]$, which suggests that while using only the current height g creates very poor trees, the current cost should be weighted higher than the lookahead value h.

We also tested our heuristics using several of the ISAC '85 benchmark circuits, interepreting the circuits as DAGs. Table 2 shows the results of this comparison. While the optimal α values are typically higher for these networks than the benchmark Bayesian networks, we see that the results are similar to the previous networks – the smallest means appear when $\alpha \in [0.1, 0.5]$. Our heuristic results in smaller trees than the standard *min-fill* algorithm, even after balancing the resulting dtree before converting to an e-tree (except for network c1908).

Table 3 and 4 show the results of using the modified *min-fill* measure as the heuristic to build e-trees for the Bayes networks and benchmark circuits, respectively. Again, the mean value of 50 trials is reported.

We can see that the results from our heuristic are generally better than those using the *min-size* heuristic as lookahead. The optimal α value appears to be lower (meaning that even less emphasis should be placed on lookahead). The results are more significant for the benchmark circuits, where the *min-fill* algorithm is superior to the dtree methods over all test networks (recall that *min-size* did not outperform the dtree methods for the *c1908* circuit.)

Conclusions and Future Work

This paper presented techniques for building good elimination trees, from which we can construct conditioning graphs. Since the time complexity of a recursive structure is a function of its height, a shallow, balanced elimination tree is desirable.

Darwiche demonstrated two methods for building balanced dtrees (Darwiche 2000; Darwiche & Hopkins 2001). We have shown in this paper a linear-time transformation to an e-tree, that guarantees the complexity of the two structures are the same. We also developed two new heuristics for directly building e-trees, extended from traditional heuristics for developing variable orderings in Bayesian networks. We show that in the example networks, the e-trees developed from these heuristics are typically smaller than those converted from dtrees.

The results obtained from our experiments show that, for

	DTree Conversion			Best-first search (values indicate α)										
	mf	*mf-bal*	*hmetis*	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Barley	21.8	19.52	15.09	20.08	**13.6**	**12.74**	**13.26**	**13.7**	**13.62**	**13.68**	**14.34**	**14.32**	**14.8**	21.8
Diabetes	60.36	23.28	18.95	54.74	**17.94**	**17.66**	**17.68**	**18.2**	**18.76**	19.86	22.38	24.78	30.48	60.36
Link	45.74	43.4	48.2	152.14	**38.46**	**37.34**	**37.64**	**37.78**	**37.2**	**37.76**	**37.84**	**38.18**	**39.72**	45.74
Mildew	13.66	12.2	11.02	14.72	**9.12**	**9.24**	**10.0**	**10.1**	**10.14**	**10.0**	**10.0**	**10.0**	**10.0**	13.66
Munin1	23.26	22.16	26.39	42.24	**18.68**	**18.22**	**18.3**	**18.76**	**20.28**	**19.46**	**19.8**	**20.0**	**21.1**	23.26
Munin2	31.44	23.76	26.16	80.8	**15.92**	**16.0**	**16.46**	**16.92**	**17.38**	**17.3**	**18.46**	**19.16**	24.6	31.44
Munin3	26.82	21.02	24.48	68.42	**16.2**	**16.34**	**16.96**	**17.0**	**18.0**	**18.84**	**18.7**	**19.5**	21.42	26.82
Munin4	27.38	22.3	28.78	80.72	**17.08**	**17.0**	**17.02**	**17.64**	**18.22**	**18.0**	**18.7**	**20.8**	**21.74**	27.38
Pigs	26.06	24.6	24.23	51.04	**19.38**	**19.58**	**19.92**	**20.24**	**19.86**	**20.5**	**20.62**	**21.46**	**22.44**	26.06
Water	15.82	15.82	16.0	**20.12**	**15.0**	**15.0**	**15.0**	**15.0**	**15.0**	**15.0**	**15.0**	**15.0**	**15.0**	15.82

Table 3: Heights of constructed e-trees on repository Bayesian networks using the modified *min-fill* heuristic for lookahead.

	DTree Conversion			Best-first search (values indicate α)										
	mf	*mf-bal*	*hmetis*	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
c432	41.72	41.4	47.2	76.4	**38.54**	**37.86**	**37.22**	**37.74**	**38.0**	**38.24**	**39.0**	**39.0**	**39.0**	41.72
c499	47.92	42.12	40.67	78.38	**38.92**	**38.2**	**38.12**	**35.9**	**37.6**	**36.0**	**36.0**	**36.0**	**40.0**	47.92
c880	55.9	51.92	54.35	121.58	**48.58**	**47.04**	**45.32**	**44.0**	**44.38**	**45.0**	**45.0**	**45.12**	**45.74**	55.9
c1355	50.38	50.38	45.78	125.56	45.8	**44.22**	**45.38**	**40.98**	**40.5**	**39.0**	**39.0**	**39.0**	**43.0**	50.38
c1908	74.12	72.48	84.8	195.46	88.14	82.98	80.06	76.24	**69.72**	**70.44**	**70.04**	**70.7**	**68.5**	74.12

Table 4: Heights of constructed e-trees on ISAC '85 benchmark circuits, also using *min-fill* for lookahead.

recursive decompositions in which the time complexity is a function of height (i.e., little space for caching), the proposed heuristics are actually preferable to other methods for construction. This applies not only to elimination trees, but would also apply to dtrees as well.

If intermediate computations are cached (Darwiche 2000), then the time complexity of recursive decompositions becomes a strict function of the width of the variable ordering. In this case, a more appropriate strategy is to minimize the width of the variable ordering, rather than the height of the tree. However, for mixed models with partial caching, a mix of both would possibly be advantageous, where the complexity is not necessarily a function of height. Further research is needed to determine such a strategy.

Recall that ties in the estimation of the height of an e-tree were broken arbitrarily, and the average of 50 runs was reported. Different choices resulted in a difference of height that in some cases exceeded 3 variables. This effect was specially dramatic early in the construction. This suggests that a more careful measure might be a dramatic improvement on the heuristics based on *min-size* or *min-fill*.

Acknowledgements

The authors gratefully acknowledge the support of NSERC.

References

Darwiche, A., and Hopkins, M. 2001. Using recursive decomposition to construct elimination orders, jointrees and dtrees. In *Trends in Artificial Intelligence, Lecture notes in AI, 2143*. Springer-Verlag. 180–191.

Darwiche, A., and Provan, G. 1996. Query dags: A practical paradigm for implementing belief network inference. In *Proceedings of the 12th Annual Conference on Uncertainty in Artificial Intelligence (UAI-96)*, 203–210. San Francisco, CA: Morgan Kaufmann Publishers.

Darwiche, A. 2000. Recursive Conditioning: Any-space conditioning algorithm with treewidth-bounded complexity. *Artificial Intelligence* 5–41.

Dechter, R. 1999. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence* 113(1-2):41–85.

Grant, K., and Horsch, M. 2005. Conditioning Graphs: Practical Structures for Inference in Bayesian Networks. In *Proceedings of the The 18th Australian Joint Conference on Artificial Intelligence*, 49–59.

Huang, C., and Darwiche, A. 1996. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning* 15(3):225–263.

Kjaerulff, U. 1990. Triangulation of graphs - algorithms giving small total state space. Technical report, Dept. of Mathematics and Computer Science, Strandvejan, DK 9000 Aalborg, Denmark.

Lauritzen, S., and Spiegelhalter, D. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society* 50:157–224.

Miller, G. L., and Reif, J. 1985. Parallel tree contraction and its application. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, 478–489.

Monti, S., and Cooper, G. F. 1996. Bounded recursive decomposition: a search-based method for belief-network inference under limited resources. *Int. J. Approx. Reasoning* 15(1):49–75.

Zhang, N., and Poole, D. 1994. A Simple Approach to Bayesian Network Computations. In *Proc. of the Tenth Canadian Conference on Artificial Intelligence*, 171–178.