Incremental Thin Junction Trees for Dynamic Bayesian networks

Frank Hutter
Computer Science Dept.
Darmstadt University of Technology
D-64293 Darmstadt, Germany
email@fhutter.de

Brenda Ng
Div. of Eng. and Appl. Sciences
Harvard University
Cambridge, MA 02138, USA
bmng@eecs.harvard.edu

Richard Dearden
RIACS / NASA Ames
Research Center
Moffett Field, CA 94035, USA
dearden@email.arc.nasa.gov

August 2, 2004

Abstract

In this paper, we study the relationship between the thin junction tree filter (TJTF) [Pas03] and the Boyen-Koller (BK) algorithm [BK98a] for approximate inference in discrete dynamic Bayesian networks. First, we review the TJTF for discrete networks and cast the BK algorithm as a special case of TJTF. Then, we employ a TJTF to automatically compute conditionally independent clusters for the BK algorithm. Theoretical work by Boyen and Koller [BK99] showed that using conditionally independent clusters strongly improves BK's error bounds, and we demonstrate that the theoretical results carry over to practice. We achieve a contract anytime algorithm which is superior to BK with marginally independent clusters and faster than TJTF in its general form.

1 Introduction

The junction tree [LS88, JLO90, CDLS99] is amongst the most widely used tools for exact inference in graphical models. It can be used for a variety of tasks, from simultaneously computing all posterior family marginals to generating the M most probable configurations [CDLS99]. However, since its space and time complexity are exponential in the induced width of the graphical model's independence graph, it is not applicable for larger problems. Bucket elimination [Dec96], another widely used framework for exact inference has long overcome this problem by introducing approximation schemes, such as Mini-Buckets [DR03]. Only recently, this has been achived for the Junction Tree by the development of the Thin Junction Tree (TJT) [BJ02] and the Thin Junction Tree Filter (TJTF) [Pas03]. Here, we employ TJTFs for exact or approximate inference in static and dynamic discrete Bayesian networks. We incrementally build a thin junction tree Υ with a user-defined bound s on size and thus bounded space and time complexity. If Υ's size remains smaller than s at all times, our version performs exact inference. However, when the introduction of a new variable into Υ would render it larger than s,

Υ is thinned by splitting large clusters into two smaller ones connected by a new separator. Since both new clusters may be subsumed by other clusters in Υ and can thus be removed, significant size reductions are possible, often with only minimal approximation error [Kjæ94].

Previous approximate junction tree algorithms [JA90, Kjæ94] build an exact junction tree Υ and if Υ is too big approximate it afterwards. This has a severe drawback as, due to space and time constraints, it might simply not be possible to build the exact Υ in the first place. Our TJTF implementation easily deals with this problem by allowing approximations in the construction phase if necessary to ensure computational feasibility. Thus, the TJT never gets too large. Another drawback of existing approaches for approximate junction trees is that they operate on fixed, static Bayesian networks, and are not flexible enough to incorporate new variables. To see the importance of this, imagine an additional sensor is installed in a complex system; one would like to simply add a variable and a potential for its CPT into the junction tree Υ instead of rebuilding Υ's whole structure [Dra95]. TJTFs can easily handle this kind of probability distributions with dynamically changing domains. The same applies for the case of dynamic Bayesian networks.

In this paper, we deal with the problem of filtering in discrete dynamic Bayesian networks (DBNs). For this problem, the Boyen-Koller (BK) algorithm [BK98a] is a very prominent approach. It works by projecting the belief state at every time step onto a set of marginally independent clusters of variables and has a bounded expected error at all times. Unfortunately, the actual error bounds are quite loose and in practice the chosen set of clusters determines the actual approximation error. There is theoretical work yielding tighter error bounds for conditionally independent sets of clusters [BK99], but until now it was neither clear how to choose these conditionally independent clusters nor whether the tighter (yet still loose) theoretical bounds carry over to improved results in practice.

Using our TJTF implementation, we automatically detect sets of conditionally independent clusters $\mathcal{C}$ for subsequent use in BK. Each set of clus-

ters $\mathcal{C}$ induces a proto-junction tree $\Psi_\mathcal{C}$ whose size $S(\Psi_\mathcal{C})$ dominates BK's space and time complexity. We find conditionally independent sets of clusters $\mathcal{C}$ with $S(\Psi_\mathcal{C}) \leq s$ yielding low approximation error and use them in BK; this way, we achieve a contract anytime algorithm *conditional BK with automatic clustering (s)*, short *CBK-AC(s)*. Compared to the manually determined marginally and conditionally independent sets of clusters suggested in [BK98a], CBK-AC(s) yields about a tenth of BK's error in the same computation time. Automatically found clusterings can thus outperform manually determined ones. This can on the one hand considerably ease the usage of conditional BK for researchers and on the other hand also enable intelligent agents to automatically improve their lower-level inference.

The rest of this paper is organized as follows. We start with preliminaries in Section 2, also reviewing junction trees and thin junction trees. Section 3 then presents the TJTF for discrete domains. Section 4 introduces the conditional BK algorithm, views it as a special case of TJTF and shows how to automatically compute conditionally independent sets of clusters for it. The CBK-AC(s) algorithm is also detailed in this section and experiments in Section 5 show its clear superiority to standard BK with the best manually determined clusters used by Boyen and Koller [BK98a]. We conclude the paper in Section 6 and list many possible extensions of this work.

2 Preliminaries

A *Bayesian network* B is a pair $\langle \mathcal{G}, \Pi \rangle$, where the independence graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed acyclic graph (DAG) in which each node $V \in \mathcal{V}$ represents a random variable and each edge $E = (U, V) \in \mathcal{E}$ defines a direct dependence of variable V on variable U. The set $pa(V) = \{U \in \mathcal{V} | (U, V) \in \mathcal{E}\}$ of parents of a random variable V is the set of variables V directly depends on; and the *family* $\mathcal{F}_V = \{V\} \cup pa(V)$ of variable V is formed by itself and its parents. For every variable $V \in \mathcal{V}$, there is a conditional probability distribution (CPD) $\pi_V \in \Pi$ that defines the probability of V taking on one of

its values given the values of its parents: $\pi_V = P(V|pa(V))$. The semantics of a Bayesian network $\langle \mathcal{G}, \Pi \rangle$ with $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is that it specifies a joint probability distribution ϕ over its variables $\mathcal{V}$ in factored form: $\phi = P(\mathcal{V}) = \prod_{V \in \mathcal{V}} \pi_V = \phi_{\mathcal{F}_V}$.

In this paper, we refer to conditional and marginal probability distributions over sets of variables $\mathbf{X}$ as *potentials* $\phi_{\mathbf{X}}$ on $\mathbf{X}$; each conditional probability distribution $\pi_V = P(V|pa(V))$ corresponds to a potential $\phi_{\mathcal{F}_V}$ on $V's$ family variables.

In *discrete* Bayesian networks each random variable $V \in \mathcal{V}$ has a finite domain D_V and the *size* $S(\phi_{\mathbf{X}})$ of a potential $\phi_{\mathbf{X}}$ is the product of the domain sizes $|D_X|$ of every variable $X \in \mathbf{X}$:

$$S(\phi_{\mathbf{X}}) = \prod_{X \in \mathbf{X}} |D_X|.$$

A *dynamic Bayesian network (DBN)* compactly represents a dynamically changing joint probability distribution over a set of random variables $\mathbf{X}$. It is a pair $\langle B_0, B_{ts} \rangle$, where B_0 is a Bayesian network specifying the prior distribution $\phi_{\mathbf{X_0}}$ over the set of variables $\mathbf{X}$ at time step 0; and B_{ts} is the *time-slice* Bayesian network over a subset of the variables $\mathbf{X_t} \cup \mathbf{X_{t+1}}$, specifying the evolution dynamics of the variables. The *filtering* problem in DBNs is, given evidence $\mathbf{e_{1:t}}$ up to time step t, to compute the *belief state* $P(\mathbf{X_t}|\mathbf{e_{1:t}})$.[1]

2.1 Junction Trees

The *junction tree* [LS88, JLO90, CDLS99] is a very prominent and widely used secondary structure for inference in graphical models. A junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ is a tree structure where each node $C \in \mathcal{C}$ represents a *cluster* of variables and each edge $S \in \mathcal{S}$ connecting two neighbouring clusters represents a *separator* between them. Each cluster $C \in \mathcal{C}$ has an associated set of variables V_C, and a cluster potential ϕ_C; analogously, each separator $S \in \mathcal{S}$

[1] We use uppercase for random variables and lowercase for variable instantiations. Bold face is used for sets of variables.

has a set of variables V_S and a separator potential ϕ_S. If a separator S_k connects clusters C_i and C_j in the junction tree, its associated set of variables is the intersection of C_i's and C_j's variables: $V_{S_k} = V_{C_i} \cap V_{C_j}$.

A defining characteristic of junction trees is the *running intersection property*: the set of variables V_{C_k} of any cluster C_k on the path between two clusters C_i and C_j in Υ is a superset of $V_{C_i} \cap V_{C_j}$. A junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ is called *consistent* if for arbitrary sets of variables $\mathbf{X}$, the marginals over $\mathbf{X}$ in arbitrary clusters C_i and C_j with $\mathbf{X} \subseteq V_{C_i}, V_{C_j}$ coincide up to a normalization constant:

$$\sum_{V_{C_i} \setminus \mathbf{X}} \phi_{C_i} \propto \sum_{V_{C_j} \setminus \mathbf{X}} \phi_{C_j}.$$

A junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ is said to be *normalized* iff all its potentials are normalized, i.e. $\forall C \in \mathcal{C}. \sum_C \phi_C = 1$ and $\forall S \in \mathcal{S}. \sum_S \phi_S = 1$. The *joint system belief* ϕ_Υ of $\Upsilon = (\mathcal{C}, \mathcal{S})$ is the joint distribution over all variables in Υ:

$$\phi_\Upsilon = \frac{\prod_{C \in \mathcal{C}} \phi_C}{\prod_{S \in \mathcal{S}} \phi_S}.$$

Constructing a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ for a Bayesian network $B = \langle \mathcal{G}, \Pi \rangle$ with $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is usually done in several steps (see [HD96] for a good overview). First, the junction tree is constructed qualitatively: the clusters $\mathcal{C}$ are formed by the maximal cliques of the network's moralized and triangulated independence graph $\mathcal{G}$, and the separators $\mathcal{S}$ are chosen to form a maximum spanning tree, where maximal is defined in terms of number of variables in the separator domains. After Υ's graphical structure is determined, a number of quantitative operations can be performed on it (for details, see, e.g. [JLO90, CDLS99, HD96]):

Initialization of $\Upsilon = (\mathcal{C}, \mathcal{S})$ means to initialize all entries in Υ's cluster potentials $\phi_C, C \in \mathcal{C}$, and separator potentials $\phi_S, S \in \mathcal{S}$, to unity.

Multiplication of Υ by a potential $\phi_\mathbf{X}$ over variables $\mathbf{X}$ means to identify some cluster $C \in \mathcal{C}$ with $\mathbf{X} \subseteq V_C$ and multiply ϕ_C by $\phi_\mathbf{X}$ (this assumes that such a cluster exists).

Division of Υ by a potential $\phi_{\mathbf{X}}$ is defined analogously to multiplication.

To put evidence e into Υ means to set all entries in all potentials of Υ to zero which do not agree with **e**.

Calibration of Υ performs a local message passing between clusters which renders Υ consistent.

Normalization of Υ normalizes every potential in Υ to sum to one.

Marginalizing $\Upsilon = (\mathcal{C}, \mathcal{S})$ to a set of variables X, later in this paper abbreviated by $marg(\Upsilon, \mathbf{X})$, means to identify a cluster $C \in \mathcal{C}$ of the consistent, normalized junction tree Υ with $\mathbf{X} \subseteq V_C$ and return its marginal $\sum_{V_C \setminus \mathbf{X}} \phi_C$ on $\mathbf{X}$ (this again assumes that such a cluster exists).

In inference for a Bayesian network $B = \langle \mathcal{G}, \Pi \rangle$ with $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, these operations are used as follows. First, the graphical structure of the junction tree Υ is determined as sketched out above. Upon initialization, the family potentials $\phi_{\mathcal{F}_V}$ for every variable $V \in \mathcal{V}$ are multiplied into Υ. This yields a joint system belief of $\phi_\Upsilon = \prod_{V \in \mathcal{V}} \phi_{\mathcal{F}_V} = P(\mathcal{V})$. Then, the available evidence **e** is put into Υ, followed by calibration and normalization. This yields the joint system belief $\phi_\Upsilon = P(\mathcal{V}|\mathbf{e})$. Afterwards, all posterior marginals $P(\mathbf{X}|\mathbf{e})$ over subsets $\mathbf{X}$ of cluster domains can be queried from Υ by simple marginalization. The correctness of this approach is guaranteed by the following theorem (this is a standard result, see, e.g. [JLO90]).

Theorem 2.1 (Correct marginals). *Marginalization of a consistent, normalized junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ to a subset $\mathbf{X}$ of any set of variables V_C associated with a cluster $C \in \mathcal{C}$, yields the joint system belief ϕ_Υ marginalized to $\mathbf{X}$:*

$$\mathrm{marg}(\Upsilon, \mathbf{X}) = \sum_{\mathcal{V} \setminus \mathbf{X}} \phi_\Upsilon.$$

Proof. Let C_i be an arbitrary cluster with $\mathbf{X} \subseteq V_{C_i}$. By definition, consistency and normality,

$$marg(\Upsilon, \mathbf{X}) = \sum_{V_{C_i} \setminus \mathbf{X}} \phi_{C_i}.$$

Because of distributivity, the marginal of the joint system belief is

$$\sum_{\mathcal{V} \setminus \mathbf{X}} \phi_\Upsilon = \frac{\prod_{C \in \mathcal{C}} \sum_{V_C \setminus \mathbf{X}} \phi_C}{\prod_{S \in \mathcal{S}} \sum_{V_S \setminus \mathbf{X}} \phi_S} = \sum_{V_{C_i} \setminus \mathbf{X}} \phi_{C_i} \times \alpha$$

with

$$\alpha = \frac{\prod_{C \in \mathcal{C} \setminus \{C_i\}} \sum_{V_C \setminus \mathbf{X}} \phi_C}{\prod_{S \in \mathcal{S}} \sum_{V_S \setminus \mathbf{X}} \phi_S}.$$

Consider a cluster C_j in $\mathcal{C} \setminus \{C_i\}$ and the adjacent separator S in the direction of C_i in the tree structure of Υ. Then, $V_{C_j} \cap \mathbf{X} = V_S \cap \mathbf{X}$. This is the case because due to $\mathbf{X} \subseteq V_{C_i}$ and the running intersection property, $x \in V_{C_j} \cap \mathbf{X}$ implies $x \in V_{C_i} \cap V_{C_j}$, which in turn implies $x \in V_S$. Because of this, as well as consistency and normality, $\sum_{V_C \setminus \mathbf{X}} \phi_{C_j} = \sum_{V_S \setminus \mathbf{X}} \phi_S$. Since there is exactly one such separator $S \in \mathcal{S}$ for each cluster $C_j \in \mathcal{C} \setminus \{C_i\}$, all terms cancel out and $\alpha = 1$. □

Theorem 2.1 may look trivial, but in Section 3 when we change the junction tree's qualitative structure by adding and marginalizing clusters, it will allow us to focus solely on the joint system belief ϕ_Υ for proving correctness of the posterior marginals $P(\mathbf{X}|\mathbf{e})$.

The task of computing posterior marginals is only detailed here as one example for the many applications the junction tree has. Compared to other exact inference algorithms like bucket elimination [Dec96], the junction tree's main advantage is that, once it is built, it can simultaneously compute all family marginals. It can also be used for the task of computing the M most probable assignments to variables in Bayesian networks [CDLS99, YW04].

2.2 Thin Junction Trees

The junction tree is a very general and efficient tool for exact computations in graphical models. Most operations in a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ have linear time complexity in Υ's size $S(\Upsilon)$, but unfortunately this size often grows prohibitively large. It is defined as the sum of the sizes of its potentials:

$$S(\Upsilon) = \sum_{\mathbf{X} \in \mathcal{C} \cup \mathcal{S}} S(\phi_{\mathbf{X}}).$$

$S(\Upsilon)$ is exponential in the induced width of the underlying graphical model which quickly renders the junction tree approach inapplicable for larger problems. The concept of *thin junction trees (TJTs)* simply deals with this problem by bounding $S(\Upsilon)$. The term originates from [BJ02], where Bach and Jordan learn the best junction tree in a restricted subset of junction trees with bounded cluster size. Since the maximal cluster size often dominates the overall junction tree size, this approach promises to yield small junction trees for efficient inference.

Another approach to manage large junction trees is to approximate them after their construction. Soon after the introduction of junction trees [LS88], Jensen [JA90] introduced such an approximation scheme that reduces the junction tree size by setting the k lowest values in each potential to zero. In 1994, Kjærulff [Kjæ94] introduced an approximation scheme for Bayesian networks that removed edges from the network's moralized, triangulated graph. He showed that this edge removal relates to splitting single large clusters of the associated junction tree into two smaller clusters connected by a separator while preserving the rest of the junction tree's structure. Since one or even both of the resulting smaller clusters may be subsumed by neighbouring clusters in the junction tree and can thus be removed, substantial size reductions can be achieved [Kjæ94].

Paskin employed thin junction trees as the state space representation in his thin junction tree filter (TJTF) for the domain of Simultaneous Localization and Mapping (SLAM) [Pas03]. In the purely continuous problem of SLAM, the cluster size is only cubic in the number of continuous variables and

the approximation error computes quite efficiently as a simple function of the covariances of the exact and approximate distributions [Pas03]. Moreover, the SLAM domain is further constrained since only two variables become coupled by each measurement. For these reasons, the TJTF is very efficient in the SLAM domain. Here, we study its application for discrete problems.

The approach we employ to reduce the size of a junction tree Υ is based on Kjærulff's [Kjæ94] work. We extend this to the case of dynamically changing probability distributions by directly operating on the junction tree itself. We iteratively compute the set $\Delta(\Upsilon)$ of possible splits of clusters in Υ and perform one possible split at a time until some termination criterion is reached.

Definition 2.2 (Possible splits). The set of *possible splits* $\Delta(\Upsilon)$ for a junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ consists of unordered pairs of variables that share exactly one cluster ($\exists!$ denotes "there exists exactly one"):

$$\Delta(\Upsilon) = \{\{u, v\} | \exists! C \in \mathcal{C}. \{u, v\} \subseteq V_C\}.$$

To perform a split $\{u, v\} \in \Delta(\Upsilon)$ **on** $\Upsilon = (\mathcal{C}, \mathcal{S})$ means to remove the unique cluster $C \in \mathcal{C}$ with $\{u, v\} \subseteq V_C$ from the consistent junction tree Υ, but add two new clusters C_u and C_v with $V_{C_u} = V_C \backslash \{v\}$ and $V_{C_v} = V_C \backslash \{u\}$; and also add a new separator S with $V_S = V_C \setminus \{u, v\}$, connecting C_u and C_v. Separators from neighbouring clusters D of C are bend to C_u if $u \in V_D$ and to C_v otherwise. The new clusters' potentials are $\phi_{C_u} = \sum_v \phi_C$ and $\phi_{C_v} = \sum_u \phi_C$, the new separator's potential is $\phi_S = \sum_{u,v} \phi_C$. Both new clusters are then checked for subsumption by their neighbouring clusters (and may often be removed rightaway). We will also refer to this operation as *thinning* the junction tree.[2]

We now define KL divergence, the error measure we use to judge about the quality of a possible split $\{u, v\} \in \Delta(\Upsilon)$.

[2]Note that possible splits only include splits of single clusters with n variables into two clusters with $n - 1$ variables each. In general, we may want to consider splits that result in smaller clusters.

Definition 2.3 (KL divergence [CT91]). The *Kullback Leibler divergence*, short KL divergence, $D(p\|q)$ between two discrete probability distributions $p(x)$ and $q(x)$ with equal domain $\mathcal{X}$ is defined as

$$D(p\|q) = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)}.$$

To quantify the error introduced by a possible split $\{u, v\} \in \Delta(\Upsilon)$, we use the KL divergence $D(\phi_\Upsilon, \phi_{\Upsilon'})$ between the joint system beliefs ϕ_Υ and ϕ'_Υ represented by the consistent and normalized junction trees Υ before the split and Υ' after the split, respectively. Kjærulff [Kjæ94] proves three very important facts for efficient approximations in junction trees: the KL divergence introduced by splitting clusters can be computed locally, it is additive, and splitting clusters preserves consistency.

Theorem 2.4 (Locality of KL divergence [Kjæ94]). *Let Υ' be the result of performing a split $\{u, v\}$ in a consistent and normalized junction tree Υ, and let C be the unique cluster in Υ containing u and v. Further, let C_u and C_v be the new clusters in Υ' generated when splitting C and let S be their new shared separator. Then, the KL divergence between ϕ_Υ and $\phi_{\Upsilon'}$ can be computed locally:*

$$D(\phi_\Upsilon\|\phi_{\Upsilon'}) = D(\phi_C\|\frac{\phi_{C_u} \times \phi_{C_v}}{\phi_S}).$$

Theorem 2.5 (Additivity of KL divergence [Kjæ94]). *The KL divergence introduced by performing a set $\{\{u_1, v_1\}, \ldots, \{u_n, v_n\}\}$ of possible splits of different clusters adds up, i.e. let Υ_0 be a junction tree and let Υ_i, $i \in \{1, \ldots, n\}$, be Υ_0 after performing splits $\{\{u_1, v_1\}, \ldots, \{u_i, v_i\}\}$. Then, $D(\phi_{\Upsilon_0}\|\phi_{\Upsilon_n}) = \sum_{j=0,\ldots,n-1} D(\phi_{\Upsilon_j}\|\phi_{\Upsilon_{j+1}})$.*

Lemma 2.6 (Conservation of consistency [Kjæ94]). *The junction tree Υ' obtained by splitting a cluster C in a consistent junction tree Υ is consistent.*

Locality of KL divergence enables us to efficiently compute the error for each possible split $\{u, v\} \in \Delta(\Upsilon)$ on a single cluster potential instead

of on the whole joint system belief ϕ_Υ (which would require summing over exponentially many values). Additivity, the fact that splitting one of Υ's clusters does not affect other clusters of Υ, and conservation of consistency together further enable a very efficient caching scheme. Let the best possible split $\{u, v\} \in \Delta(\Upsilon)$ split cluster $C \in \mathcal{C}$ into C_u and C_v, yielding the new junction tree Υ'. Then, in order to compute the new set of possible splits $\Delta(\Upsilon')$, we merely have to remove any splits $\{u', v'\}$ from $\Delta(\Upsilon)$ which also split cluster C, and instead put in the new possible splits of C_u and C_v or their respective subsuming clusters in Υ'.

So far, there are two approaches which approximate junction trees to achieve computational feasibility [JA90, Kjæ94]. However, both have the significant disadvantage that they first build an exact junction tree and approximate it afterwards; this is problematic if the exact junction tree is too large to compute in the first place. In the next section, we introduce a set of operations that allows for incremental construction of the junction tree to overcome this problem.

3 The Thin Junction Tree Filter for Discrete networks

In this section, we present the thin junction tree filter (TJTF) [Pas03] for exact or approximate inference in static and dynamic discrete Bayesian networks. By incrementally building junction trees, the framework can represent probability distributions over both a fixed set and a changing set of dynamic variables. Variables and their conditional probability tables can be introduced into the junction tree Υ, they can be summed out of Υ and Υ's size is kept as low as possible by removing non-maximal clusters. For cases where TJTF without approximations is not feasible, we split clusters for computational efficiency as demonstrated in the last section.

The following operations build on the ones introduced by Draper [Dra95] for incremental construction of qualitative junction trees, but extend them to

a quantitative treatment, handling cluster and separator potentials as well.

To introduce new cliques $\mathcal{X}$ into $\Upsilon = (\mathcal{C}, \mathcal{S})$ means to build a new Junction Tree $\Upsilon' = (\mathcal{C}', \mathcal{S}')$ such that for each $C \in \mathcal{C}$ there is a $C' \in \mathcal{C}'$ with $V_C \subseteq V_{C'}$, and that for each new clique $\mathbf{X} \in \mathcal{X}$ there is a $C' \in \mathcal{C}'$ with $\mathbf{X} \subseteq V_{C'}$. Υ' is then initialized, multiplied by the potentials ϕ_C, $C \in \mathcal{C}$, and divided by ϕ_S, $S \in \mathcal{S}$.

To introduce a variable V into $\Upsilon = (\mathcal{C}, \mathcal{S})$ whose parents are all contained in Υ is shorthand for introducing a clique for V's family $\mathcal{F}_V$ and multiplying V's family potential ϕ_V into Υ.

A subsumption check of $C \in \mathcal{C}$ by $D \in \mathcal{C}$ in $\Upsilon = (\mathcal{C}, \mathcal{S})$ means to check whether $V_C \subseteq V_D$. If this is the case, we say C is *subsumed* by D. C is then removed from Υ along with the separator S between C and D; all other separators adjacent to C are bend over to D. D's potential ϕ_D is multiplied by ϕ_C and divided by the separator potential ϕ_S:

$$\phi_D \leftarrow \frac{\phi_D \times \phi_C}{\phi_S}.$$

To merge a connected set of clusters $\mathcal{C}' \subseteq \mathcal{C}$ in $\Upsilon = (\mathcal{C}, \mathcal{S})$ means to delete each $C \in \mathcal{C}'$ and the separators $S \in \mathcal{S}'$ connecting one $C_i \in \mathcal{C}'$ to another $C_j \in \mathcal{C}'$ from Υ and instead introduce one new cluster M with $V_M = \bigcup_{C \in \mathcal{C}'} V_C$ and

$$\phi_M = \frac{\prod_{C \in \mathcal{C}'} \phi_C}{\prod_{S \in \mathcal{S}'} \phi_S}.$$

All neighbours of M in Υ are then checked for subsumption by M in Υ.

To sum a variable V out of $\Upsilon = (\mathcal{C}, \mathcal{S})$ means to merge the connected set of clusters $\{C \in \mathcal{C} | V \in V_C\}$ in Υ into a new cluster M and afterwards marginalize V out of M's potential: $\phi_M \leftarrow \sum_V \phi_M$. M is then checked for subsumption by its neighbours in Υ.

Theorem 2.1 states that even when performing complex changes in the qualitative junction tree structure, for correctness of the posterior marginals we only need to consider the joint system belief ϕ_Υ. We now show how ϕ_Υ behaves under the new set of operations.

Lemma 3.1 (Uniformly extended joint system belief). *Introduction of new cliques $\mathcal{X}$ extends the joint system belief ϕ_Υ of a junction tree uniformly to the new variables in $\mathcal{X}$.*

Proof. When introducing new cliques $\mathcal{X}$ into $\Upsilon = (\mathcal{C}, \mathcal{S})$, we build a new junction tree Υ' to include variables from the new cliques and initialize all potentials with uniform probability distributions. Then, we multiply each of the old potentials $\phi_C, C \in \mathcal{C}$ into Υ'. Consequently, the marginal over the pre-existing variables does not change and the marginal over the newly introduced variables is uniform. Thus, the new joint system belief ϕ'_Υ is the old one ϕ_Υ uniformly extended over the new variables in the cliques $\mathcal{X}$. □

Lemma 3.2 (Unchanged joint system belief). *Subsumption checks and merging of a connected set of clusters in a junction tree Υ do not change Υ's joint system belief ϕ_Υ.*

Proof. Let $\Upsilon' = (\mathcal{C}', \mathcal{S}')$ be the result of performing a subsumption check or a merging of a connected set of clusters in the junction tree Υ. For each cluster $C \in \mathcal{C}$ that is removed from $\Upsilon = (\mathcal{C}, \mathcal{S})$, there is some other cluster $D \in \mathcal{C}$ which is not removed and whose cluster potential ϕ_D is multiplied by ϕ_C. Analogously, for every separator $S \in \mathcal{S}$ that is removed there is a $D \in \mathcal{C}$ whose cluster potential ϕ_D is divided by ϕ_S. Thus, $\phi'_\Upsilon = \prod_{C \in \mathcal{C}'} \phi_C / \prod_{S \in \mathcal{S}'} \phi_S = \prod_{C \in \mathcal{C}} \phi_C / \prod_{S \in \mathcal{S}} \phi_S = \phi_\Upsilon$. □

Lemma 3.3 (Marginalization of ϕ_Υ). *Summing a variable V out of a junction tree Υ marginalizes Υ's joint system belief ϕ_Υ over V: $\phi_{\Upsilon'} = \sum_V \phi_\Upsilon$.*

Proof. Let $\Upsilon'' = (\mathcal{C}'', \mathcal{S}'')$ denote the result of merging the clusters containing V in $\Upsilon = (\mathcal{C}, \mathcal{S})$. Because of Lemma 3.2, $\phi_\Upsilon = \phi_{\Upsilon''}$. Let M denote the

merged cluster, which is now the unique cluster holding V and let Υ' denote the result of marginalizing V out of Υ''.

$$\begin{aligned}\phi_{\Upsilon'} &= (\textstyle\sum_V \phi_M) \times \frac{\prod_{C \in \mathcal{C}'' \setminus \{M\}} \phi_C}{\prod_{S \in \mathcal{S}''} \phi_S} \\ &= \textstyle\sum_V \frac{\prod_{C \in \mathcal{C}''} \phi_C}{\prod_{S \in \mathcal{S}''} \phi_S} \\ &= \textstyle\sum_V \phi_{\Upsilon''} = \sum_V \phi_\Upsilon.\end{aligned}$$

□

Using a TJTF with all of the introduced operations but thinning results in a special case of TJTF we refer to as the *junction tree filter (JTF)*. JTFs perform exact inference in (static or dynamic) Bayesian networks. We introduce new variables, their cliques and conditional probability tables, and marginalize others out while keeping the size down by means of subsumptions. The correctness of this approach is guaranteed by the following theorem.

Corrolary 3.1 (Correctness of JTFs). *Consider an initially empty JTF Υ, into which a set of variables $\mathbf{X}$ is incrementally introduced and out of which a subset $\mathbf{Y} \subseteq \mathbf{X}$ of variables is summed out. If in this process no thinning has taken place, Υ's joint system belief is the product of the family potentials $\phi_{\mathcal{F}_V}$ marginalized over $\mathbf{Y}$:*

$$\phi_\Upsilon = \sum_{\mathbf{Y}} \prod_{V \in \mathbf{X}} \phi_{\mathcal{F}_V}.$$

Proof. Before any variable is introduced, the joint system belief is 1. The introduction of a variable V into Υ first introduces the clique $\mathcal{F}_V$ and then multiplies in the potential $\phi_{\mathcal{F}_V}$. Due to Lemma 3.1 and since multiplication of Υ by $\phi_{\mathbf{X}}$ multiplies ϕ_Υ by $\phi_{\mathbf{X}}$, this results in a new joint system belief $\phi'_\Upsilon = \phi_\Upsilon \times \phi_{\mathcal{F}_V}$. Due to Lemma 3.3, summing an arbitrary variable V out of Υ yields a marginalized joint system belief $\phi_{\Upsilon'} = \sum_V \phi_\Upsilon$. After introducing the variables $V \in \mathbf{X}$ and summing out the variables $V \in \mathbf{Y}$, the joint system belief is thus $\phi_\Upsilon = \sum_{\mathbf{Y}} \prod_{V \in \mathbf{X}} \phi_{\mathcal{F}_V}$. □

Using JTFs, we can often keep an exact probability distribution over the variables which are of interest in the given situation. Variables which are

of no more interest are marginalized out. An approach quite related to this is the frontier algorithm for DBNs [Zwe96]. This method keeps a joint distribution over a set of variables, the *frontier*, which separates the past from the future and is "swept" over the DBN forwards in time (and backwards for smoothing). It multiplies in new variables and marginalizes out old variables as soon as all their children are contained in the frontier. Our JTF approach, however, does not necessarily build only a single large cluster over the whole frontier when applied to DBNs. Rather, if the joint distribution factorizes, it may take advantage of this by employing smaller clusters while still performing exact inference. Nevertheless, without approximations, JTFs would still often produce clusters that span the whole frontier, which may lead to prohibitive sizes of the junction tree.

JTFs are correct but just like the standard junction tree approach they are computationally infeasible for large problems. The following lemma and theorem guarantee that we can always deal with this by performing approximations.

Lemma 3.4 (Maximality of clusters). *All clusters in a TJTF are maximal.*

Proof. A newly constructed junction tree Υ formed when introducing cliques has maximal clusters. This property is conserved by the standard junction tree operations in Section 2.1 which do not change Υ's qualitative structure. Possible non-maximal clusters, which are left by merging or splitting clusters, or summing variables out, are always checked for subsumption. They are subsumed iff they are non-maximal. □

Corrolary 3.2 (Existence of splits). *Every TJTF $\Upsilon = (\mathcal{C}, \mathcal{S})$ with at least one cluster C with $|V_C| > 1$ has at least one possible split.*

Proof by construction. The clusters $C \in \mathcal{C}$ with $|V_C| > 1$ induce a subforest Υ' of Υ. Consider an arbitrary leaf cluster C in Υ'. If C has no neighbour in Υ', any pair of different variables $\{u, v\}$ with $u, v \in V_C$ is a possible

split in Υ since due to the running intersection property u and v can only appear together in C. If C has a neighbour D in Υ', due to C's maximality (Lemma 3.4), there is at least one variable $v \in V_C$ which does not appear in D. Variable v then forms a possible split $\{v, x\}$ of Υ with every other $x \in V_C$. □

Since all clusters in a TJTF are maximal, iterated approximations eventually lead to a size reduction and terminate when every cluster $C \in \mathcal{C}$ has only one variable left.

TJTF can be applied in either static or dynamic Bayesian networks. We expect the algorithm's potential for DBNs to be higher, and we discuss this in detail in the next section. For static models, TJTFs provide an alternative to Kjærulff's method of edge removal [Kjæ94]. His approach is a special case of the one we suggest here, where the specialization is to restrict all approximations to happen after all variables have been introduced. This is very reasonable because it enables the most informed approximation, but if the exact junction tree is prohibitively large it is infeasible. To counter this, Kjærulff suggests first approximating the cluster potentials using sampling techniques. TJTFs are feasible without this secondary approximation technique. Given a bound on space complexity, we can topologically introduce variables and thin the junction tree Υ once the junction tree Υ' resulting from the introduction of a new variable would grow too large. Approximate junction trees obtained like this can yield approximations to all problems junction trees apply for, such as the simultaneous approximation of all posterior marginals and the approximation of the M most probable explanations in a Bayesian network [CDLS99, YW04] [3].

[3] One possible application of computing the M-MPEs in a thinned junction tree is in providing a set of very good initial solutions for stochastic local search algorithms for the 1-MPE problem. It is widely known that stochastic local search algorithms generally benefit from good initial solutions quite significantly; for the 1-MPE this has been demonstrated convincingly in [KD99].

4 TJTF and the Boyen-Koller algorithm

In this section, we describe the Boyen-Koller algorithm for approximate inference in DBNs, show how it can be viewed as a special case of the TJTF and based on this notion compute conditionally independent clusters for it.

4.1 The Boyen Koller algorithm

The Boyen-Koller (BK) algorithm [BK98a] is a commonly used for approximate inference in DBNs. As Boyen and Koller [BK98a] demonstrate, although a complex system's evolution dynamics can be represented in compact form as a DBN, in general all conditional independencies amongst variables in a time slice are lost when marginalizing out variables from previous time slices. In exact inference, this quickly results in a prohibitively large cluster whose variables are the so-called *canonical* variables $\mathbf{X}_t$, variables in time slice t which are parents of variables in slice $t+1$. The BK algorithm's approach is to approximate the belief state $P(\mathbf{X_t}|\mathbf{e_{1:t}})$ at every time step t by a product of marginal probability distributions over subsets of variables:

$$P(\mathbf{X_t}|\mathbf{e_{1:t}}) \approx \prod_{C_t \in \mathcal{C}_t} P(C_t|\mathbf{e_{1:t}}),$$

where $\mathcal{C}_t$ is a partition of $\mathbf{X}_t$.

The intuition behind this is that the DBN describes a complex system which consists of only weakly interacting subsystems represented by the clusters $C_t \in \mathcal{C}_t$; and that the error induced by ignoring covariances between variables in different subsystems is small due to the only weak interactions amongst subsystems. The accuracy of BK is strongly sensitive to the partition $\mathcal{C}$ of canonical variables and finding a good $\mathcal{C}$ requires deep domain knowledge and is often manually done by experts. On the positive side, due to the stochasticity of the process, the individual errors introduced at each time step decay exponentially fast over time, such that the expected error of BK is bounded at all times [BK98a]. Unfortunately, the error bounds are quite loose, but in [BK99], Boyen and Koller significantly improve them

for conditionally independent clusters. Their characterization of weak and sparse interactions between subsystems of complex systems [BK99] provides a strong theoretical foundation for conditionally independent sets of clusters. However, implementation issues and empirical results have been disregarded so far. For clarity of presentation, we detail the BK algorithm using conditionally independent sets of clusters $\mathcal{C}$ (*conditional BK($\mathcal{C}$)*, short *CBK($\mathcal{C}$)*). The implementation of BK with a conditionally independent set of clusters adds just a few extra lines of code to the marginally independent case and is explained in a short footnote in [BK98b]. Unfortunately, this explanation is misleading.[4] For this reason and due to the importance of the algorithm, we present the conditional BK algorithm at some length. With the term *clustering*, we refer to a marginally or conditionally independent set of clusters $\mathcal{C}$ to use in CBK($\mathcal{C}$). The approximate belief state in CBK($\mathcal{C}$) is

$$P(\mathbf{X_t}|\mathbf{e_{1:t}}) \approx \frac{\prod_{C_t \in \mathcal{C}_t} P(C_t|\mathbf{e_{1:t}})}{\prod_{S_t \in \mathcal{S}_t} P(S_t|\mathbf{e_{1:t}})}, \tag{1}$$

where $S_t \in \mathcal{S}_t$ are the separators in a junction tree with clusters $\mathcal{C}_t$.

As described in [BK98a], the propagation of a belief state from one time step to the next in a DBN $\langle B_0, B_{ts} \rangle$ can be implemented very efficiently in (conditional) BK. For a given clustering $\mathcal{C}$, Boyen and Koller suggest building a so-called *proto-junction tree* Ψ. We can easily explain the construction of Ψ with the TJTF framework. For this purpose, let $\mathcal{F}_{\text{ts}}$ denote the set of families in B_{ts} and Φ_{ts} denote their associated family potentials.

[4] Boyen and Koller (see footnote on page 22 of [BK98b]) describe a different denominator. For a set of clusters $C_1, \ldots, C_n$, instead of dividing by the potentials over the separators $S_1, \ldots, S_{n-1}$ of a junction tree formed by $C_1, \ldots C_n$, they divide by potentials over all non-empty intersections $V_{C_i} \cap V_{C_j}, i < j$. For the one conditionally independent set of clusters A-B-C-D-E, C-D-E-F-G, G-H presented in [BK98b], this agrees with our version (both would choose the approximation $(\phi_{ABCDE} \times \phi_{CDEFG} \times \phi_{GH})/(\phi_{CDE} \times \phi_G)$), but in general it yields wrong results. Suppose a system with the variables A,B,C,D,E,F decomposes into the conditionally independent clusters A-B-C-D, B-C-D-E, and C-D-E-F. The joint probability ϕ_{ABCDEF} is then $(\phi_{ABCD} \times \phi_{BCDE} \times \phi_{CDEF})/(\phi_{BCD} \times \phi_{CDE})$, which we would choose as an approximation, while their version would additionally divide by ϕ_{CD}.

To build a proto-junction tree $\Psi_{\mathcal{C}}$ for a clustering $\mathcal{C}$ and a time slice Bayesian network B_{ts} means to create a new empty junction tree and introduce cliques $\{V_C | C \in \mathcal{C}_t \cup \mathcal{C}_{t+1}\} \cup \mathcal{F}_{\text{ts}}$ into it. Unless $\Psi_{\mathcal{C}}$ is only built qualitatively, $\Psi_{\mathcal{C}}$ is initialized and the time slice family potentials Φ_{ts} are multiplied into it. We call $\Psi_{\mathcal{C}}$ the *proto-junction tree induced by* $\mathcal{C}$.

Figure 1 shows CBK($\mathcal{C}$) for filtering in a DBN given a stream of observations $\mathbf{e_{1,2,\ldots}}$ and using the clustering $\mathcal{C}$. First, the separators $\mathcal{S}$ for clustering $\mathcal{C}$ are computed. Then, the proto-junction tree $\Psi_{\mathcal{C}}$ is constructed and the priors $\phi_{C_0} = P(V_{C_0})$ and $\phi_{S_0} = P(V_{S_0})$ are computed for $C_0 \in \mathcal{C}_0$ and $S_0 \in \mathcal{S}_0$, respectively (lines 02–05). At every time step t, a temporary copy Ψ of $\Psi_{\mathcal{C}}$ is multiplied by the cluster potentials ϕ_{C_t}, $C_t \in \mathcal{C}_t$, and divided by the separator potentials ϕ_{S_t}, $S_t \in \mathcal{S}_t$ (lines 07–09). New evidence $\mathbf{e_{t+1}}$ is then introduced and Ψ is calibrated (lines 10–11). The next time step's cluster and separator potentials $\phi_{C_{t+1}}, C_{t+1} \in \mathcal{C}_{t+1}$, and $\phi_{S_{t+1}}, S_{t+1} \in \mathcal{S}_{t+1}$, can then be retrieved by querying Ψ (lines 12–13). As [BK98a] notes, this is a straight-forward extension of the standard BK algorithm with marginally independent clusterings. The only additional work we need to do for the conditionally independent clustering is to compute the separators (line 01), initialize, input the old separator potentials and query the new ones (lines 05, 09, and 13). The separators do not impose new constraints on the structure of Ψ since for every separator $S \in \mathcal{S}$, there is a cluster $C \in \mathcal{C}$ with $V_S \subseteq V_C$.

The space requirement of CBK($\mathcal{C}$) is linear in its induced proto-junction tree size $S(\Psi_{\mathcal{C}})$. The dominating factor in time complexity is $\Psi_{\mathcal{C}}$'s calibration, which is also linear in $S(\Psi_{\mathcal{C}})$. Thus, for efficient inference our aim is to find a clustering that induces a small proto-junction tree. However, the finer a clustering we choose, the bigger the error introduced by the approximation in Equation 1; we face a tradeoff of time and space versus approximation error. In the next two sections, we show how to automatically find a very good clustering with bounded induced proto-junction tree size based on the TJTF framework.

Algorithm CBK($\mathcal{C}, \langle B_0, B_{\text{ts}} \rangle, \mathbf{e_{1,2,\ldots}}$)
Input. Clustering $\mathcal{C}$, DBN $\langle B_0, B_{\text{ts}} \rangle$, continuous stream of observations $\mathbf{e_{1,2,\ldots}}$
Precondition. $\mathbf{X} \subseteq \bigcup_{C \in \mathcal{C}} V_C$ for canonical vars $\mathbf{X}$.

Output. Filter marginals $P(V_{C_t} | \mathbf{e_{1:t}}) = \phi_{C_t}$ for all clusters $C \in \mathcal{C}$ and $t = 0, 1, \ldots$
%===== Compute separators.
01. $(\mathcal{C}, \mathcal{S}) \leftarrow$ Connect clusters $\mathcal{C}$ into a qualitative JT.
%===== Init Ψ_{Proto} and potentials for $\mathcal{C}_0$ and $\mathcal{S}_0$.
02. Build proto-junction tree $\Psi_{\mathcal{C}}$ for $\mathcal{C}$ and B_{ts}.
03 $\Upsilon_0 \leftarrow$ Build junction tree for B_0
04. For each $C \in \mathcal{C}$, $\phi_{C_0} = \text{marg}(\Upsilon_0, V_{C_0})$
05. For each $S \in \mathcal{S}$, $\phi_{S_0} = \text{marg}(\Upsilon_0, V_{S_0})$
%===== Do propagation for each time step.
06. **for** $t = 0, 1, \ldots$ **do**
07. $\quad \Psi \leftarrow \Psi_{\mathcal{C}}$
08. $\quad$ **for each** $C \in \mathcal{C}$ **do** Multiply Ψ by ϕ_{C_t}
09. $\quad$ **for each** $S \in \mathcal{S}$ **do** Divide Ψ by ϕ_{S_t}
10. $\quad$ Insert new evidence $\mathbf{e_{t+1}}$ into Ψ.
11. $\quad$ Calibrate Ψ.
12. $\quad$ **for each** $C \in \mathcal{C}$ **do** $\phi_{C_{t+1}} \leftarrow \text{marg}(\Psi, V_{C_{t+1}})$.
13. $\quad$ **for each** $S \in \mathcal{S}$ **do** $\phi_{S_{t+1}} \leftarrow \text{marg}(\Psi, V_{S_{t+1}})$.
14. **end**

Figure 1: The conditional BK algorithm for approximate inference in DBNs. Step 1 can be achieved by any maximum-spanning-tree algorithm, see e.g. [HD96]. For a marginally independent clustering $\mathcal{C}$, we can omit lines 1, 5, 9, and 13.

4.2 A TJTF algorithm generalizing CBK

In this section, we describe a TJTF algorithm that generalizes the CBK algorithm introduced in the last section. At every time step t, this new algorithm adaptively chooses a set of clusters $\mathcal{C}_t$ and separators $\mathcal{S}_t$, or, to be explicit, a junction tree $\Upsilon_t = (\mathcal{C}_t, \mathcal{S}_t)$, to approximate the belief state $P(\mathbf{X_t}|\mathbf{e_{1:t}})$. For this, it takes into account the observations $\mathbf{e_{1:t}}$ which enables easy detection of and reaction to special events such as sparse interactions between subsystems [BK99].

For the filtering task in DBNs, we start at time step t with a junction tree $\hat{\Upsilon}_t$, representing the filter estimate at t: $\phi_{\hat{\Upsilon}_t} = \hat{P}(\mathbf{X_t}|\mathbf{e_{1:t}})$. This junction tree is then thinned down to a junction tree $\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)$ by repeatedly splitting clusters until some termination criterion is satisfied.

Then, we introduce the variables $\mathbf{X_{t+1}}$ into $\tilde{\Upsilon}_t$ and also insert the evidence $\mathbf{e_{t+1}}$. This yields a junction tree $\hat{\Upsilon}_{t+1}$ with $\phi_{\hat{\Upsilon}_{t+1}} = \hat{P}(\mathbf{X_t}, \mathbf{X_{t+1}}|\mathbf{e_{1:t+1}})$. Normally, we could keep some of the variables $\mathbf{X_t}$ in Υ_{t+1}, possibly achieving smaller size $S(\Upsilon_{t+1})$ by exploiting conditional independence[5]. However, in the spirit of CBK, here we sum all variables $\mathbf{X_t}$ out of $\hat{\Upsilon}_{t+1}$, yielding the next time step's filter estimate $\phi_{\hat{\Upsilon}_{t+1}} = \hat{P}(\mathbf{X_{t+1}}|\mathbf{e_{1:t+1}})$.

At every time step t, the junction tree $\hat{\Upsilon}_t$ which represents the filter estimate is approximated by a junction tree $\tilde{\Upsilon}_t$. However, we are not constrained to use the same approximation $\tilde{\Upsilon}$ at every time step. This feature allows us to react optimally to the given situation, thinning $\hat{\Upsilon}_t$ differently conditioned on the last belief state $\tilde{\Upsilon}_{t-1}$ and especially on the new observation $\mathbf{e_t}$. Knowing the new evidence, we can exactly compute the error introduced by different thinnings and perform the best one in the given situation. From a theoretical point of view, the complete analysis of BK [BK98a, BK99] applies to this TJTF algorithm since it is done on a step-by-step base. In fact, it is usually

[5]This would conflict with the general approach of keeping the number of variables in the belief state representation as small as possible and would rather seek to minimize the size s of the belief state representation; s might indeed become smaller if some past variables decoupling many present variables do not have to be marginalized out.

hard in BK to determine a bound on the individual error ϵ introduced at each time step [BK98a]; in our setting, we exactly know all the individual errors and thus get this bound for free as the highest error introduced over all time steps so far. The biggest problems with the stated algorithm is its high computational cost which is due to the costly evaluation of many possible thinnings at each time step. It may also in part be an artifact of our non-optimized Matlab implementation.[6]

The stated TJTF algorithm generalizes CBK since using a static clustering, or namely, the same $\tilde{\Upsilon} = (\tilde{\mathcal{C}}, \tilde{\mathcal{S}})$ at every time step yields exactly CBK($\tilde{\mathcal{C}}$). The major advantage of CBK, however, is speed. CBK does not need to perform a large number of possibly costly splits of clusters but directly collects the cluster and separator marginals. Also, computing a new approximation at every time step does not allow for the usage of the highly efficient proto-junction tree implementation. For this reason, in our sample application, we only perform the TJTF approach in an off-line fashion on a training sequence $\bar{\mathbf{e}}_{\mathbf{1}:\mathbf{T}_{\text{train}}}$ in order to determine good clusterings $\tilde{\mathcal{C}}$ to be subsequently used as fixed clusterings in the faster CBK($\tilde{\mathcal{C}}$) algorithm.

4.3 Computing well-performing clusterings for CBK

Boyen and Koller [BK98a] clearly state that the choice of clustering significantly influences both BK's runtime and approximation error. However, they suggest to simply use a clustering reflecting the subsystems of the complex system represented by the DBN at hand. This approach requires domain knowledge and significant trial-and-error on the researcher's side in order to determine a clustering that is computationally feasible and results in fairly accurate inference. Here, we present a simple algorithm which automatically computes clusterings for CBK that perform orders of magnitude better than the best ones determined manually by Boyen and Koller [BK98a]. The fact that we automatically find better clusterings than by hand will ease the use

[6] In particular, significant speedups may be gained by using more vectorized code than is currently done.

of CBK for new DBNs and also enable intelligent agents to improve their lower level inference engines automatically.

For a given bound s, we focus on the problem of finding a clustering $\mathcal{C}$ with a low approximation error of CBK($\mathcal{C}$) and an induced proto-junction tree size $S(\Psi_{\mathcal{C}}) \leq s$. In Figure 2, we present pseudo code for our algorithm *AC(s)*, that **A**utomatically finds a good **C**lustering $\mathcal{C}$ with $S(\Upsilon_{\mathcal{C}}) \leq s$ for subsequent use in CBK($\mathcal{C}$). As $S(\Upsilon_{\mathcal{C}})$ dominates CBK's space and time complexity, this yields a contract anytime algorithm *CBK-AC(s)* in combination with *CBK($\mathcal{C}$)*.

For finding a good clustering, AC(s) first samples a training sequence $\bar{\mathbf{e}}_{\mathbf{1}:\mathbf{T}_{\text{train}}}$ of evidence from the DBN $\langle B_0, B_{\text{ts}} \rangle$ and builds a TJTF for the prior belief state $P(\mathbf{X_0})$ represented by B_0 (see Figure 2, line 01–02). For every time step t from 0 to T_{train}, we iteratively and greedily split clusters of the TJTF $\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)$ until the induced proto-junction tree size $S(\Psi_{\tilde{\mathcal{C}}_t})$ is smaller than or equal to s (lines 04–09)[7]. After the thinning process for one time step is complete, we move on to the next time step by introducing the variables $\mathbf{X_{t+1}}$ into the TJTF, summing out old variables $\mathbf{X_t}$, introducing the new evidence $\mathbf{e_{t+1}}$ and calibrating (lines 10–13). Upon completion of the given number of time steps, we sample a new sequence of observations $\bar{\mathbf{e}}_{\mathbf{1}:\mathbf{T}_{\text{choose}}}$ and evaluate the clusterings $\tilde{\mathcal{C}}_1, \ldots, \tilde{\mathcal{C}}_{T_{\text{train}}}$ on this one. The clustering $\tilde{\mathcal{C}}_t$ with best accuracy of CBK($\tilde{\mathcal{C}}_t$) is returned (line 16).

The search for a good clustering $\tilde{\mathcal{C}}$ in AC(s) is performed offline. Since the online filtering algorithm CBK($\tilde{\mathcal{C}}$) can be used for arbitrarily long sequences, the time for searching a good clustering quickly amortizes. Precomputing various clusterings for different maximal proto-junction tree sizes s yields a contract anytime algorithm for online filtering, *conditional BK with automatic clustering*, *CBK-AC(s)*. In CBK-AC(s), an increase in computational

[7]The quality of a possible split $\{u, v\} \in \Delta(\Upsilon)$ has to be determined by some heuristic, and in this case we choose the split $\{u, v\}$ maximizing the ratio of reduction in induced proto-junction tree size and approximation error, as measured by KL divergence. This greedy heuristic is motivated by the objective to achieve large size reductions while introducing only a small approximation error.

Algorithm AC(s, $\langle B_0, B_{\text{ts}} \rangle$)

Input. DBN $\langle B_0, B_{\text{ts}} \rangle$, bound s on $S(\Upsilon)$

Output. Clustering $\tilde{\mathcal{C}}$ for use in CBK

%===== Sample training evidence and init TJTF.

01. Sample $\bar{\mathbf{e}}_{\mathbf{1}:\mathbf{T}_{\text{train}}}$ from $\langle B_0, B_{\text{ts}} \rangle$.
02. $\hat{\Upsilon}_0 \leftarrow$ Build TJTF for B_0

%===== Thin until induced proto-jt small enough.

03. **for** t = 0 to T_{train}
04. $\quad (\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)) \leftarrow \hat{\Upsilon}_t$
05. $\quad \Psi_{\tilde{\mathcal{C}}_t} \leftarrow$ Qualitative proto-junction tree$(\tilde{\mathcal{C}}_t, B_{\text{ts}})$
06. $\quad$ **while**$(S(\Psi_{\tilde{\mathcal{C}}_t}) >$ bound**) do**
07. $\quad\quad (\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)) \leftarrow$Thin $\tilde{\Upsilon}_t$
08. $\quad\quad \Psi_{\tilde{\mathcal{C}}_t} \leftarrow$ Qual. proto-junction tree $(\tilde{\mathcal{C}}_t, B_{\text{ts}})$
09. $\quad$ **end**

%===== Move on to next time step.

10. $\quad \hat{\Upsilon}_{t+1} \leftarrow$Introduce new variables $\mathbf{X_{t+1}}$ into $\hat{\Upsilon}_t$
11. $\quad$ Sum out variables $\mathbf{X_t}$ from $\hat{\Upsilon}_{t+1}$
12. $\quad$ Introduce evidence $\mathbf{e_{t+1}}$ into $\hat{\Upsilon}_{t+1}$
13. $\quad$ Calibrate $\hat{\Upsilon}_t$
14. **end**

%===== Choose best performing clustering.

15. Sample $\bar{\mathbf{e}}_{\mathbf{1}:\mathbf{T}_{\text{choose}}}$ from $\langle B_0, B_{\text{ts}} \rangle$.
16. $\tilde{\mathcal{C}} \leftarrow \tilde{\mathcal{C}}_t \in \{\tilde{\mathcal{C}}_1, \ldots, \tilde{\mathcal{C}}_{T_{\text{train}}}\}$ with best accuracy of the filter marginals for CBK$(\tilde{\mathcal{C}}_t, \langle B_0, B_{\text{ts}} \rangle, \bar{\mathbf{e}}_{\mathbf{1}:\mathbf{T}_{\text{choose}}})$.

Figure 2: Finding good conditionally independent cliques for BK by using the TJTF framework. In our experiments, the parameters T_{train} and T_{choose} are set to 10 and 50, respectively.

resources allows the usage of a larger proto-junction tree which in turn lets us use a set of precomputed clusters with higher induced proto-junction tree size resulting in lower approximation error. However, there remains a problem for large DBNs $\langle B_0, B_{\text{ts}} \rangle$. If B_{ts} has many densely connected variables and thus high induced width, even the induced proto-junction tree Ψ_{ff} for the fully factored clustering (where every canonical variable has its own cluster) might be infeasibly large [MW01]. Here, our anytime approach reaches a natural border since we cannot thin the fully factored clustering any further. CBK-AC(s) in its current version is thus not completely anytime yet. We would like to address this in future work by thinning the proto-junction tree $\Psi_{\mathcal{C}}$ directly instead of merely thinning the junction tree $\Upsilon = (\mathcal{C}, \mathcal{S})$ inducing it.[8] This way, we can achieve a true anytime algorithm.

5 Experiments

In this section, we report results of CBK-AC(s) for the task of filtering in the BAT [FHKR95] and WATER [JKOP89] DBNs, the same networks as used by Boyen and Koller [BK98a]. For each of the networks, we compare CBK-AC(s) with BK using the best manually determined sets of clusters reported in [BK98a].[9] As an overall error measure, we employ the maximal error in the filter marginals at each time step.

For each DBN, we ran CBK-AC(s) with a number of different bounds s on induced proto-junction tree size. For each clustering $\mathcal{C}$ thus obtained for

[8] In our approach so far, we only approximate the filter estimate $P(\mathbf{X_t}|\mathbf{e_{1:t}})$ and use the exact transition function $P(\mathbf{X_{t+1}}|\mathbf{X_t})$, no matter how beneficial it might be in terms of induced proto-junction tree size to also approximate the transition function. By thinning the proto-junction tree directly, we could achieve an approximation of both the filter estimate *and* the transition function at once. The algorithm would then be free to choose the most promising approximations.

[9] For much standard functionality, we used Kevin Murphy's Bayes Net Toolbox [Mur01]. Our experiments are done on a 1.2GHz Pentium laptop with 768MB Ram running Windows XP.

Algorithm	PJT Size	Error	Time
BK ff	6752	1.30e-3	8.37
BK bk	60660	4.22e-4	10.06
BK cond	464942	1.91e-4	33.98
BK exact	5004482	0	-
CBK-AC(6752)	6752	1.30e-3	8.27
CBK-AC(10000)	7892	8.43e-4	8.40
CBK-AC(15000)	13074	2.17e-4	8.61
CBK-AC(20000)	16434	1.68e-4	8.81
CBK-AC(30000)	28786	1.67e-4	9.98
CBK-AC(50000)	43794	8.15e-5	10.46
CBK-AC(80000)	74226	8.27e-5	12.34
CBK-AC(200000)	179698	4.09e-5	17.74
CBK-AC(400000)	382322	6.82e-6	34.69

Table 1: Size of the induced proto-junction tree, average error of the filter marginals and wall clock time for various online algorithms on the WATER network. BK exact ran out of memory on our machines.

the WATER network, Figure 3 shows $\mathcal{C}$'s induced proto-junction tree size $S(\Psi_{\mathcal{C}})$ and the average errors of CBK-AC($\mathcal{C}$)'s filter marginals. We compare this to standard BK with fully factored (ff) clusters, the best manually determined marginally independent set of clusters (bk), and the conditionally independent set of clusters (cond) suggested in [BK98a]. BK with just one single large cluster (exact) would yield a proto-junction tree size of 5004482; however, our machines ran out of memory when using this clustering.

In Table 1, we list the induced proto-junction tree size, average error of the filter marginals, and run time for the algorithms on the WATER network. The induced proto-junction tree size s of a clustering $\mathcal{C}$ indeed dominates the time complexity of CBK($\mathcal{C}$) for larger s.

The development of the errors in CBK-AC($\mathcal{C}$)'s filter marginals over time is shown in Figure 4. It follows the same pattern as observed in [BK98a]: the errors are quite low most of the time, with a few spikes. The variance

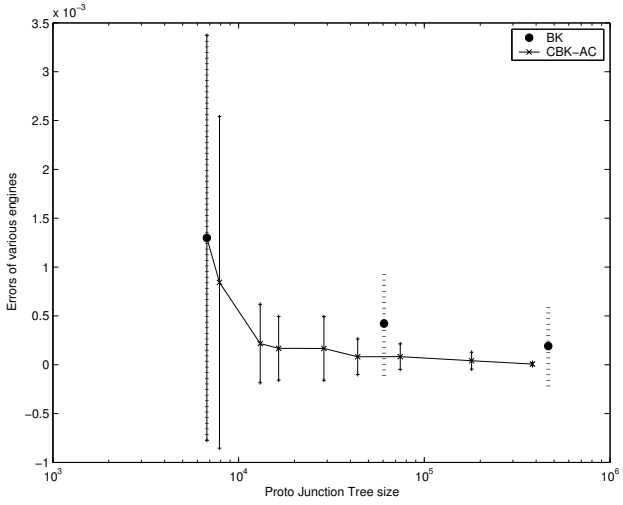

Figure 3: Error in the filter estimates of CBK-AC(s) and BK on the WATER network, averaged over 100 time steps. BK clusterings: ff, A-B, C-D-E-F, G-H (bk) and the conditionally independent clustering A-B-C-D-E, C-D-E-F-G, G-H (cond), as suggested in [BK98a]. BK-exact has induced proto-junction tree size 5004482 and runs out of memory.

of the marginal errors is quite high for all clusterings, not only for BK-ff. The advantage of using the automatically found clusters of CBK-AC(s) becomes obvious in Figure 5, where we plot the ratios of BK-bk error and BK-cond error to the error of the respective CBK-AC(s) version with next lower induced proto-junction tree size s (and thus, at most equal complexity). On average, it yields approximation errors which are between one and two orders of magnitude better than the standard BK variants with comparable complexity.

For the BAT network, we achieve very similar results. Table 2 shows for each clustering $\mathcal{C}$ used in CBK-AC the size of the induced proto-junction tree, average error of the filter marginals, and the run time. Note that the clustering found by CBK-AC(28554) induces a proto-junction tree that is smaller than Ψ_{ff}, the proto-junction tree induced by the fully factored clus-

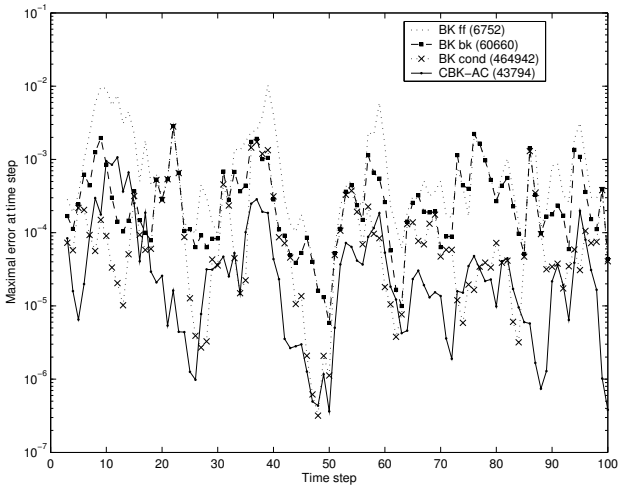

Figure 4: Maximal error in the filter marginal per time step for CBK-AC and BK for the WATER domain. The number in parenthesis gives the size of the clustering's induced proto-junction tree.

tering. This is possible because we do not build the minimal junction tree for each clustering. In constrast, for all clusterings we employ standard software [Mur01] implementing the min-weight heuristic (see e.g. [HD96]) for the $\mathcal{NP}$-hard problem of finding the minimal junction tree. We can luck out and find a clustering which yields a small proto-junction tree with this heuristic (even smaller than the one found for the fully factored clustering), but we can as well have bad luck and construct a very poor junction tree. CBK-AC(28554) obviously yields much lower error than BK-ff. Ψ_{ff} necessarily places many variables in shared clusters; approximating their joint distribution by a product of marginals as done in BK-ff can thus not yield any advantage in computation time or space, but merely introduces an unnecessary error. The superiority of our automatically found clusters is again demonstrated in Figure 7; there, CBK-AC(s) yields much lower error than BK-bk with a smaller induced proto-junction tree size s.

Algorithm	PJT Size	Error	Time
BK ff	28554	5.23e-4	15.20
BK bk	49099	2.98e-5	14.96
BK exact	1465342	0	-
CBK-AC(28554)	28473	4.74e-5	16.48
CBK-AC(35000)	33513	1.75e-5	15.24
CBK-AC(40000)	37977	5.55e-7	15.90
CBK-AC(45000)	37977	5.55e-7	15.90
CBK-AC(55000)	50091	3.45e-7	16.65
CBK-AC(75000)	62331	3.47e-7	17.05
CBK-AC(100000)	62331	3.37e-7	17.05
CBK-AC(200000)	130878	1.76e-7	20.23
CBK-AC(400000)	130878	1.76e-7	20.23

Table 2: Size of the induced proto-junction tree, average error of the filter marginals and wall clock time for various online algorithms on the BAT network. BK exact ran out of memory on our machines.

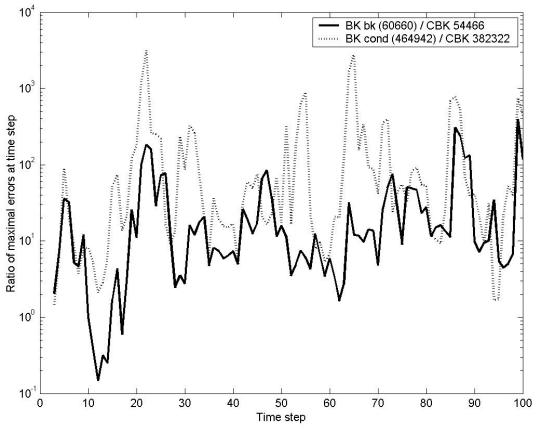

Figure 5: Ratios of average marginal errors of BK and CBK-AC(s) with smaller proto-junction tree on the Water network.

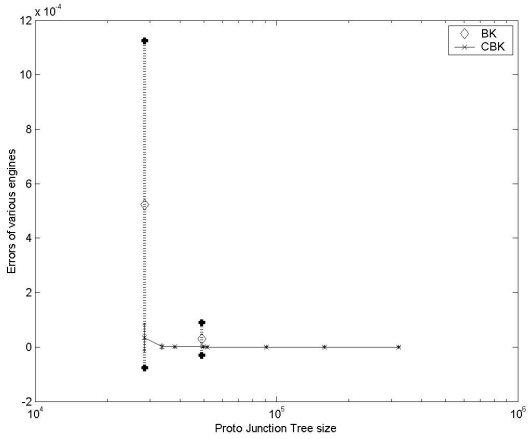

Figure 6: Average error in the filter estimates of CBK-AC(s) and BK on the BAT network. BK-exact would induce a proto-junction tree of size 1465342 and ran out of memory on our machines.

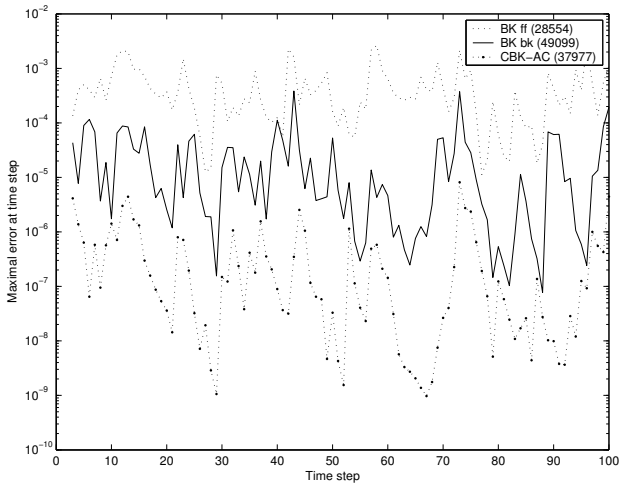

Figure 7: Maximal error in the filter marginal per time step for CBK-AC and BK for the BAT domain. The number in parenthesis gives the size of the clustering's induced proto-junction tree.

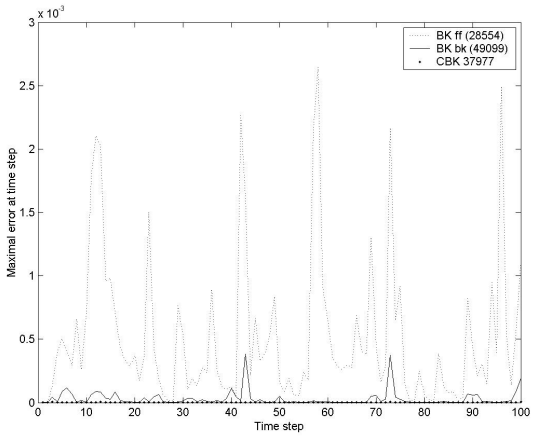

Figure 8: Same as Figure 7, but with nonlogarithmic scale. Here, the error peaks for BK-ff can be seen very clearly.

6 Conclusions and Future work

In this paper, we studied the relationship between the thin junction tree filter (TJTF) for inference in static and dynamic Bayesian networks (DBNs) on the one hand and the Boyen-Koller algorithm for approximate inference in DBNs on the other hand. We explicitly stated the conditional Boyen-Koller algorithm (CBK) mentioned in [BK98a], showed that TJTFs generalize CBK and how they can be used to automatically find conditionally independent clusterings $\mathcal{C}$ with low error of CBK($\mathcal{C}$) and an induced proto-junction tree $\Upsilon_{\mathcal{C}}$ with bounded size $S(\Upsilon_{\mathcal{C}}) < s$. We showed that $S(\Upsilon_{\mathcal{C}})$ determines the space and time complexity of CBK($\mathcal{C}$); thus, we achieved a contract anytime algorithm CBK-AC(s) that finds a good clustering $\mathcal{C}$ with $S(\Upsilon_{\mathcal{C}}) < s$ and then performs CBK($\mathcal{C}$) online for an arbitrary amount of time steps. On the WATER and BAT DBNs, we demonstrated empirically how CBK-AC(s) shows significantly lower approximation errors than standard BK with equal computational complexity.

The TJTF framework can be applied in a number of other areas. It can e.g. be used to build approximate junction trees on large Bayesian networks. For the filtering task, TJTF can detect special circumstances such as sparse interactions between variables very naturally. We plan to develop this promising approach further; its potential is an efficient algorithm which exactly quantifies the actual error it introduces with specific approximations. In order to make this framework applicable for online inference we need to reduce its complexity. We plan research on approximation schemes to estimate the error of splitting a cluster and speed up the introduction of new variables by means of Draper's operations for incremental building of qualitative junction trees [Dra95]. Another interesting point is that the theoretical analysis of CBK [BK98a, BK99] does not apply directly to the TJTF framework if we split clusters between time steps. We conjecture that the theoretical properties should be similar, but this needs further research.

There are also many possibilities to improve and extend the CBK-AC(s) algorithm. As discussed in Section 4.3, its contract anytime property is nat-

urally limited by the size of the induced proto-junction tree Υ_{ff} of a fully factored clustering. There are DBNs for which Υ_{ff} is prohibitively large [MW01] but by building an approximate proto-junction tree we can achieve the full contract anytime property; we plan to implement this in future work

Using a matrix of proto-junction trees, it is also possible at the runtime of CBK to switch between several clusterings that have been computed offline. If the time and space constraints for online inference change, the clustering can be adapted to reflect this. Furthermore, if we can assess the performance of different clusterings in regular intervals of the online algorithm, a reactive scheme becomes possible that also detects sparse interactions [BK99] between variables and uses an appropriate clustering at any time.

We also plan to study how the approaches studied in this paper can be used for inference in hybrid DBNs and how they can be combined with other approximate inference schemes.

For hybrid DBNs with conditional Gaussian distributions, Paskin's approach [Pas03] is most promising, whereas for general hybrid DBNs combinations with particle filtering seem most promising. We plan to study an extension of the factored particle filter [NPP02] to conditionally independent clusterings, which could be determined with a procedure similar to AC(s). By employing Rao-Blackwellized particle filters [DdFMR00] or the Gaussian particle filter [HD03], factored particles also generalize to hybrid DBNs with probability distributions close to multimodal multivariate Gaussians [NPDH04]. This promising novel approach in filtering for DBNs can likely be further improved by means of conditionally independent clusterings.

7 Acknowledgements

We thank Xavier Boyen for providing the original WATER and BAT networks used in [BK98a]. We are also indebted to the anonymous reviewers for an earlier version of this paper submitted to UAI for their expedient criticism that was instrumental in gaining a better of connections to related work.

References

[BJ02] Francis R. Bach and Michael I. Jordan. Thin junction trees. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *NIPS 14*, pages 569–576. MIT Press, 2002.

[BK98a] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 33–42, 1998.

[BK98b] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. Technical report, Stanford University, USA, 1998.

[BK99] Xavier Boyen and Daphne Koller. Exploiting the architecture of dynamic systems. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 313–320, 1999.

[CDLS99] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer, 1999.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications. Wiley Interscience, 1991.

[DdFMR00] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart J. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 176–183, 2000.

[Dec96] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[DR03] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, 2003.

[Dra95] Denise Draper. Clustering without (thinking about) triangulation. In Philippe Besnard and Steve Hanks, editors, *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 125–133, 1995.

[FHKR95] Jeff Forbes, Timothy Huang, Keiji Kanazawa, and Stuart J. Russell. The batmobile: Towards a bayesian automated taxi. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1878–1885, 1995.

[HD96] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[HD03] Frank Hutter and Richard Dearden. The gaussian particle filter for diagnosis of non-linear systems. In *Proceedings of the 14th International Conference on Principles of Diagnosis(DX'03)*, pages 65–70, 2003.

[JA90] Finn V. Jensen and S. Andersen. Approximations in bayesian belief universes for knowledge based systems. Technical report, Institute of Electronic Systems, Aalborg University, Aalborg, Denmark, 1990.

[JKOP89] Finn V. Jensen, Uffe Kjærulff, Kristian G. Olesen, and Jan Pedersen. Et forprojekt til et ekspertsystem for drift af spildevandsrensning (an expert system for control of waste water treatment — a pilot project). Technical report, Judex Datasystemer A/S, Aalborg, Denmark, 1989. In Danish.

[JLO90] Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4:269–282, 1990.

[KD99] Kalev Kask and Rina Dechter. Stochastic local search for bayesian networks. In *Proc. of AISTATS-99*, 1999.

[Kjæ94] Uffe Kjærulff. Reduction of computational complexity in bayesian networks through removal of weak dependences. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 374–382, 1994.

[LS88] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[Mur01] Kevin Murphy. The bayes net toolbox for matlab. *Computing Science and Statistics*, 33, 2001.

[MW01] Kevin Murphy and Yair Weiss. The factored frontier algorithm for approximate inference in dbns. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 378–385, 2001.

[NPDH04] Brenda Ng, Avi Pfeffer, Richard Dearden, and Frank Hutter. Factored sampling for monitoring nonlinear hybrid systems with autonomous transitions. Submitted to UAI, 2004.

[NPP02] Brenda Ng, Leonid Peshkin, and Avi Pfeffer. Factored particles for scalable monitoring. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 370–377, 2002.

[Pas03] Mark A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1157–1164, 2003.

[YW04] Chen Yanover and Yair Weiss. Finding the m most probable configurations using loopy belief propagation. In *NIPS 16*. MIT Press, 2004.

[Zwe96] G. Zweig. A forward-backward algorithm for inference in bayesian networks and an empirical comparison with hmms. Master's thesis, Dept. Comp. Sci., U.C. Berkeley, 1996.