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

Abstract

We present Incremental Thin Junction Trees, a general framework for approximate inference in static and dynamic Bayesian Networks. This framework incrementally builds junction trees representing probability distributions over a dynamically changing set of variables. Variables and their conditional probability tables can be introduced into the junction tree Υ, they can be summed out of Υ and Υ can be approximated by splitting clusters for computational efficiency. As one of many possible applications of this general framework we automatically generate conditionally independent clusters for the Boyen-Koller (BK) algorithm. Theoretical work by Boyen and Koller [2] showed that using conditionally independent clusters strongly improves BK's error bounds. We show how to identify these conditionally independent clusters automatically and that the theoretical results carry over to practice. We achieve a contract anytime algorithm which is superior to BK with marginally independent clusters.

1 Introduction

The junction tree [9, 3] 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 [3]. 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. Here, we propose a generalized framework, the *Incremental Thin Junction Tree (ITJT)* for exact or approximate inference in static and dynamic Bayesian Networks. In ITJT, 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 framework performs exact inference. 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 be removed, significant size reductions are possible, often with only minimal approximation error [8].

Existing approximate junction tree algorithms [7, 8] 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 may not be possible to build the exact Υ in the first place. Our ITJT framework deals with this problem by allowing approximations in the construction phase if necessary to ensure computational feasibility. Thus, ITJTs never get too large. Another drawback of the 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 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 [4]. ITJTs can easily handle such dynamically changing domains, as well as dynamic Bayesian Networks.

As a sample application of ITJTs, we chose filtering in dynamic Bayesian Networks (DBNs). For this problem, the Boyen-Koller (BK) algorithm [1] 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 bounds are quite loose and in practice the chosen set of clusters determines the actual approximation error. There is theoretical work yielding tighter bounds for conditionally independent sets of clusters [2], but until now it was neither clear how to choose these conditionally independent clusters nor whether the theoretical bounds carry over to improved results in practice.

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

ters $\mathcal{C}$ 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 clusters suggested in [1], 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.

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 causal relationship from variable U to variable V. The set of parents of a random variable V is the set of variables V directly depends on; and V's *family* $\mathcal{F}_V$ 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: $\pi_V = P(V|pa(V))$. In this paper, we view these CDPs as *potentials* $\phi_{\mathcal{F}_V}$ on V's family variables. 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: $\phi = P(\mathcal{V}) = \prod_{V \in \mathcal{V}} \pi_V = \prod_{V \in \mathcal{V}} \phi_{\mathcal{F}_V}$.

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 $\mathbf{X}$ 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 domain we chose for demonstration of our framework is the *filtering* problem in DBNs: given evidence $\mathbf{e_{1:t}}$ up to time step t, compute the *belief state* $P(\mathbf{X_t}|\mathbf{e_{1:t}})$.[1]

2.1 Junction Trees

The *junction tree* [9, 3] is a 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 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 $S \in \mathcal{S}$ 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 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 [5] 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. [3, 5]):

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 clique $C \in \mathcal{C}$ with $\mathbf{X} \subseteq V_C$ and multiply ϕ_C by $\phi_\mathbf{X}$.

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

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

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** $\mathbf{X}$**,** 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 $\mathbf{X}$.

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,

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

the graphical structure of the junction tree Υ is determined as outlined 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 $\mathbf{e}$ 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.

Theorem 2.1 (Correct marginals).[2] *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 $\mathbf{X}$:*

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

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* simply deals with this problem by bounding $S(\Upsilon)$.

Soon after the introduction of junction trees [9], Jensen [7] introduced an approximation scheme that reduces the junction tree size by setting the k lowest values in each potential to zero. In 1994, Kjærulff [8] 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 both of the resulting smaller clusters may be subsumed by neighbouring clusters and can thus be removed, substantial size reductions can be achieved [8]. Thin junction trees have also been used by Paskin for Simultaneous Localization and Mapping (SLAM) [12]. His interesting application is specifically tailored to the SLAM domain and does not generalize to discrete DBNs. In the purely continuous problem of SLAM, the cluster size is only cubic in the number of continuous variables and the approximation error is a simple function of the covariances of the exact and approximate distributions [12]. Moreover, SLAM is further constrained since only two variables are coupled by each measurement. For these reasons, approximation is much easier in SLAM.

[2]The proofs for this and other theorems in the paper can be found in the companion technical report [6].

The approach we employ to reduce the size of a junction tree Υ is based on Kjærulff's [8] work. We extend this to the case of dynamically changing probability distributions by directly operating on the junction tree itself. We iteratively detect a set $\Delta(\Upsilon)$ of possible splits of clusters in Υ and perform one of them 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:

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

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 $V_{C_u} = V_C \setminus \{v\}$ and $V_{C_v} = V_C \setminus \{u\}$; 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 move 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 neighbours. We will also refer to this operation as *thinning* the junction tree.

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. Kjærulff [8] proves three very important facts for efficient approximations in junction trees: the KL divergence can be computed locally, it is additive, and splitting clusters preserves consistency.

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 put in the new possible splits of C_u and C_v or their respective subsuming clusters in Υ'.

There are two approaches which approximate junction trees to achieve computational feasibility [7, 8]. 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 Incremental Thin Junction Trees

Incremental Thin Junction Trees (ITJT) is a framework for exact or approximate inference in static and dynamic Bayesian Networks. By building junction trees incrementally, 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 ITJT 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 [4] 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: $\phi_D \leftarrow (\phi_D \times \phi_C)/(\phi_S)$.

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 $\phi_M = (\prod_{C \in \mathcal{C}'} \phi_C)/(\prod_{S \in \mathcal{S}'} \phi_S)$. 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 belief ϕ_Υ of a junction tree uniformly to the new variables in $\mathcal{X}$.*

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.*

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: $\phi_{\Upsilon'} = \sum_V \phi_\Upsilon$.*

Using an ITJT with all the introduced operations but thinning (we abbreviate this as *IJT*), we can 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.

Theorem 3.4 (Correctness of IJTs). *Consider an initially empty ITJT 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 $\mathbf{Y}$: $\phi_\Upsilon = \sum_{\mathbf{Y}} \prod_{V \in \mathbf{X}} \phi_{\mathcal{F}_V}$.*

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

Theorem 3.5 (Existence of splits). *Every ITJT $\Upsilon = (\mathcal{C}, \mathcal{S})$ with at least one cluster C with $|V_C| > 1$ has at least one possible split.*

Since all clusters in an ITJT are maximal [6], iterated approximations eventually lead to a size reduction and terminate when every cluster $C \in \mathcal{C}$ has only one variable left.

ITJT can be applied in either static or dynamic Bayesian Networks. The algorithm has the most potential in DBN problems, and we discuss this in detail in the next section. For static models, ITJTs provide an alternative to Kjærulff's method of edge removal [8]. His approach is a special case of ours which is achieved by first introducing all variables and afterwards performing the approximation. This enables the most informed approximation, but if the exact junction tree is prohibitively large it is infeasible. To counter this, Kjærulff suggests to first approximating the cluster potentials using sampling techniques. ITJTs are feasible without this secondary approximation technique. Given a bound on space complexity, we can incrementally introduce variables and thin the junction tree once it would grow too large. Approximate junction trees obtained like this can then be used for a variety of tasks in large problems where exact junction trees are computationally infeasible. For example, the simultaneous approximation of all posterior marginals and the approximation of the M most probable explanations (M-MPEs) in a Bayesian Network [3].

4 Boyen-Koller with conditionally independent clusters

The Boyen-Koller (BK) algorithm [1] is commonly used for approximate inference in DBNs. Its approach is to approximate the belief state $P(\mathbf{X_t}|\mathbf{e_{1:t}})$ at every step t by a product of marginal 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 is that the DBN describes a 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 $\mathcal{C}$ 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 over time so the expected error is bounded at all times [1]. Unfortunately, the error bounds are quite loose, but in [2], Boyen and Koller significantly improve them for conditionally independent clusters. While the theory for this case is very well developed [1, 2], 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($\mathcal{C}$)*). 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

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

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

As described in [1], 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 our 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.

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 $\mathcal{C}$.

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 $\mathbf{e_{1,2,\ldots}}$
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 $C \in \mathcal{C}$, ϕ_{C_0} = marg(Υ_0, V_{C_0})
05. For each $S \in \mathcal{S}$, ϕ_{S_0} = marg(Υ_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** $\phi_{C_{t+1}} \leftarrow$ marg($\Psi, V_{C_{t+1}}$).
13. $\quad$**for each** $S \in \mathcal{S}$ **do** $\phi_{S_{t+1}} \leftarrow$ marg($\Psi, V_{S_{t+1}}$).
14. **end**

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

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). Note, that this is a straight-forward extension of the BK algorithm with marginally independent clusterings [1].

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.

4.1 An ITJT algorithm generalizing CBK

It is possible to formulate an ITJT algorithm that strictly generalizes CBK by propagating an ITJT over time. At every time step, it introduces the new variables $\mathbf{X_{t+1}}$, puts in the new evidence $\mathbf{e_{t+1}}$, marginalizes out the old vari-

ables $\mathbf{X_t}$ and then approximates the remaining junction tree by splitting clusters. If the same approximation is used at every time step, this algorithm is equivalent to CBK, but in general it is much more powerful as it can adaptively choose its approximations conditioned on the observations $\mathbf{e_{1:t}}$ and thus can detect and react to special events such as sparse interactions between subsystems [2]. Due to space restrictions, we omit details; these can be found in the companion technical report [6]. In the next section, we employ a similar methodology to compute well-performing clusterings for the less flexible but faster CBK algorithm.

4.2 Computing well-performing clusterings for CBK

Boyen and Koller [1] 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 [1]. The fact that we automatically find better clusterings than by hand will ease the use 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 an ITJT 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 ITJT $\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)$ until the induced proto-junction tree size $S(\Psi_{\tilde{\mathcal{C}}_t})$ is smaller than or equal to s (lines 04–09)[3]. 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 ITJT, summing out old variables $\mathbf{X_t}$, introducing the new evidence $\mathbf{e_{t+1}}$ and calibration (lines 10–13). Upon completion of the given number of time steps, we sample a new sequence of observations 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).

[3]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 ITJT.
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 ITJT for B_0
%===== Thin until induced proto-jt small enough.
03. **for** t = 0 to T_{train}
04. $(\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)) \leftarrow \hat{\Upsilon}_t$
05. $\Psi_{\tilde{\mathcal{C}}_t} \leftarrow$ Qualitative proto-junction tree($\tilde{\mathcal{C}}_t, B_{\text{ts}}$)
06. **while(**$S(\Psi_{\tilde{\mathcal{C}}_t}) >$ bound**) do**
07. $(\tilde{\Upsilon}_t = (\tilde{\mathcal{C}}_t, \tilde{\mathcal{S}}_t)) \leftarrow$Thin $\tilde{\Upsilon}_t$
08. $\Psi_{\tilde{\mathcal{C}}_t} \leftarrow$ Qual. proto-junction tree ($\tilde{\mathcal{C}}_t, B_{\text{ts}}$)
09. **end**
%===== Move on to next time step.
10. $\hat{\Upsilon}_{t+1} \leftarrow$Introduce new variables $\mathbf{X_{t+1}}$ into $\hat{\Upsilon}_t$
11. Sum out variables $\mathbf{X_t}$ from $\hat{\Upsilon}_{t+1}$
12. Introduce evidence $\mathbf{e_{t+1}}$ into $\hat{\Upsilon}_{t+1}$
13. 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 ITJT framework. In our experiments, the parameters T_{train} and T_{choose} are set to 10 and 50, respectively.

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 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 variable has its own cluster) might be infeasibly large [11]. 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 will 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 [4]. 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 and WATER DBNs, the same networks as used by Boyen and Koller [1]. [5] For each of the networks, we compare CBK-AC(s) with BK using the best manually determined sets of clusters reported in [1] [6]. 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 the WATER network, Figure 3 (a) 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 [1]. 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 3 (b). It follows the same pattern as observed in [1]: the errors are quite low most of the time, with a few spikes. The variance of the marginal errors is quite high for all clusterings, not only for BK-ff. Although CBK-AC(s) has error spikes similar to BK, 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 clustering. This is possible because we employ standard software [10] implementing the min-weight heuristic (see e.g. [5]) for the $\mathcal{NP}$-hard problem of finding the minimal junction tree. We can thus luck out and find a clustering which yields a small proto-junction tree with this heuristic. CBK-AC(28554) also 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 3 (c); 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	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.

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: Same as Table 1, but for the BAT network.

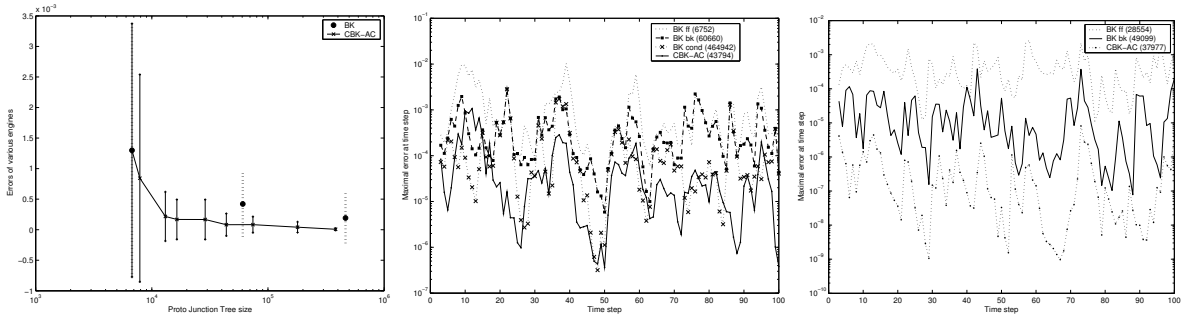
6 Conclusions and Future work

In this paper, we introduced the general framework of Incremental Thin Junction Trees (ITJTs) for exact or approximate inference in Bayesian Networks and demonstrated it for the filtering problem in DBNs. For this problem,

[4] 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 will achieve an approximation of both the filter estimate *and* the transition function at once. The algorithm will then be free to choose the most promising approximations.

[5] We thank Xavier Boyen for providing the original WATER and BAT DBNs used in [1]; a detailed description of the networks and pointers to their origin can also be found in that paper.

[6] Our code is written in Matlab and available at `http://www.fhutter.de/itjt.html`. For much standard functionality, we use Kevin Murphy's Bayes Net Toolbox [10]. Our experiments are done on a 1.2GHz Pentium laptop with 768MB Ram running Windows XP.



(a) Average error: WATER domain (b) Maximal error: WATER domain (c) Maximal error: BAT domain

Figure 3: (a) 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 [1]. BK-exact has induced proto-junction tree size 5004482 and runs out of memory.
(b and c) Maximal error in the filter marginal per time step for CBK-AC and BK for the WATER (b) and BAT (c) domains. The number in parenthesis gives the size of the clustering's induced proto-junction tree.

we explicitly stated the conditional Boyen-Koller algorithm (CBK), showed that ITJTs generalize CBK and how they can be used to automatically find conditionally independent clusterings with low error and an induced proto-junction tree with bounded size. 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 ITJT 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, we showed how it can incorporate evidence into the choice of approximation at every time step and thus adapt to the situation at hand. This enables ITJT to detect special circumstances such as sparse interactions between variables very naturally. 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 [4]. Another interesting point is that the theoretical analysis of CBK [1, 2] does not apply directly to the ITJT 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 our CBK-AC(s) algorithm. As discussed in Section 4.2, its contract anytime property is naturally 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 [11] but by building an approximate proto-junction tree we can achieve the full contract anytime property. It is also possible at runtime 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.

References

[1] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. of UAI-98*, pages 33–42, 1998.

[2] X. Boyen and D. Koller. Exploiting the architecture of dynamic systems. In *Proc. of UAI-99*, pages 313–320, 1999.

[3] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer, 1999.

[4] D. Draper. Clustering without (thinking about) triangulation. In P. Besnard and S. Hanks, editors, *Proc. of UAI-95*, pages 125–133, 1995.

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

[6] F. Hutter, B. Ng, and R. Dearden. Incremental thin junction trees for dynamic bayesian networks. Technical report, Intellectics Group, Darmstadt University of Technology, Germany, 2004. To be submitted. Preliminary version at www.fhutter.de/itjt.pdf.

[7] F. 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.

[8] U. Kjærulff. Reduction of computational complexity in bayesian networks through removal of weak dependences. In *Proc. of UAI-94*, pages 374–382, 1994.

[9] S. L. Lauritzen and D. 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.

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

[11] K. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in dbns. In *Proc. of UAI-01*, pages 378–385, 2001.

[12] M. A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *Proc. of IJCAI-03*, pages 1157–1164, 2003.