The Workshops of the Thirtieth AAAI Conference on Artificial Intelligence
Beyond NP: Technical Report WS-16-05

Exploiting the Hidden Structure of Junction Trees for MPE

Batya Kenig and Avigdor Gal
Technion, Israel Institute of Technology
{batyak@tx,avigal@ie}.technion.ac.il

Abstract

The role of decomposition-trees (also known as junction and clique trees) in probabilistic inference is widely known and has been the basis for many well known inference algorithms. Recent approaches have demonstrated that such trees have a "hidden structure", which enables the characterization of tractable problem instances as well as lead to insights that enable boosting the performance of inference algorithms. We consider the MPE problem on a Boolean formula in CNF where each literal in the formula is associated with a weight. We describe techniques for exploiting the junction-tree structure of these formulas in the context of a branch-and-bound algorithm for MPE.

Introduction

In Probabilistic Graphical Models, a *Most Probable Explanation* (MPE) is an assignment to all network variables that has a maximum probability given the evidence. The problem of finding the MPE is known to be NP-hard in general. Algorithms for inferring the exact MPE (Jensen, Lauritzen, and Olesen 1990; Lauritzen and Spiegelhalter 1988; Zhang and Poole 1996; Dechter 1999) exploit the graph topological structure and their runtime is exponential in a graph-theoretic measure called *treewidth*. Such an exponential bound implies that tractability in PGMs can only be achieved by bounding its treewidth.

The topological structure encoded in the graphical model does not capture other structural properties such as determinism and Context Specific Independence (CSI), which may enable efficient inference despite a large treewidth (Chavira and Darwiche 2008). An inference algorithm that can exploit both topological and local structure will be exponential in the network's treewidth only in the worst case. Bayesian networks can be translated to a Boolean formula in Conjunctive Normal Form (CNF) with weighted literals whose size is linear in their conditional probability tables (CPTs), making the CNF-based MPE problem strictly more general.

A recent approach, introduced by Kenig and Gal, capitalizes on the structural properties of a CNF's tree-decomposition (also known as clique-tree, junction-tree or join-tree) for performing Weighted Model Counting (WMC). In particular, the hidden structure exploited by these algorithms cannot generally be revealed by the translation process to CNF. Solving the MPE problem on a CNF is likely to be easier than solving the Weighted Model Counting (WMC) on the same instance because we may apply branch-and-bound based pruning techniques to reduce the search space. Furthermore, the refined complexity measures introduced in (Kenig and Gal 2015) and the characterizations of formulas whose weighted model count can be computed in P-Time (Capelli, Durand, and Mengel 2014; Kenig, Gal, and Strichman 2013), automatically carry over to the optimization case. These type of results fit in with efforts to bridge the gap between theoretical performance guarantees and those empirically observed by current solvers (Szeider 2011).

In this paper we extend the algorithms of (Kenig and Gal 2015) with effective branch-and-bound pruning techniques. We apply the methods for leveraging the structure of the CNF's tree-decomposition along with existing methods for exploiting CSI (Boutilier et al. 1996) in the context of a message-passing algorithm over junction-trees. We adapt proven optimizations from the CNF-SAT domain such as unit propagation and caching into our algorithm in order to create a practical tool for MPE.

CNF-trees for MPE

Propositional or Boolean variables can take two values $\{1, 0\}$. A *literal* l of a variable X is either a variable or its negation, which are denoted by $x, \bar{x}$, respectively. The variable corresponding to a literal l is denoted by $\mathrm{var}(l)$. We assume that the variables are weighted and for every variable X_j we denote by w_{x_j},$\mathrm{w}_{\bar{x}_j}$ the weights of its positive and negative literals, respectively. The weight of an assignment $\mathbf{X} = \mathbf{x}$ is denoted by $w(\mathbf{x})$.

A Boolean formula f over variables $\mathbf{X}$ maps each instantiation $\mathbf{x}$ to either *true* or *false*. We assume that $f(\mathbf{X})$ is in Conjunctive Normal Form (CNF), *i.e.*, a conjunction of *clauses*, each containing a disjunction of literals. We denote by $\phi_1, \phi_2, .., \phi_n$ the set of unique clauses in f, where every ϕ_i represents a set of literals. We assume that the formula f is simplified, meaning, for every pair of clauses $\phi_i, \phi_j \in f$, $\phi_i \nsubseteq \phi_j$. Conditioning a CNF formula f on literal l, denoted $f|l$, consists of removing the literal $\bar{l}$ from all clauses, and

dropping the clauses that contain l. Conditioning a formula on a set of literals $\gamma = \{l_1, l_2, ..., l_k\}$, denoted $f|\gamma$, amounts to conditioning it on every literal $l \in \gamma$.

We say that a variable X *affects* the formula's outcome if $f|x \neq f|\bar{x}$. We denote by $\mathrm{var}(f)$ the set of variables that affect the formula. A pair of formulae f_1, f_2 are disjoint if $\mathrm{var}(f_1) \cap \mathrm{var}(f_2) = \emptyset$. The model of f with the maximum weight (or probability) is denoted $\mathrm{MPE}(f)$.

We model the formula $f(\mathbf{X})$ as a CNF interaction graph, an undirected graphical model, $G_f(\mathbf{X}, E)$, where each variable corresponds to a node in the graph and there is an edge between pairs of variables that belong to a common clause.

Let $T_r(\mathcal{I})$ denote a rooted junction tree for $G_f(\mathbf{X}, E)$, where $\mathcal{I}$ represents the node indices, and the members of each node $i \in \mathcal{I}$ are denoted $\mathbf{X}_i$. According to the junction tree properties, each clause, $\phi_i \in f$, is associated with a node $i \in \mathcal{I}$ such that $\mathrm{var}(\phi_i) \subseteq \mathbf{X}_i$. This node-clause relationship is reflected in the tree by attaching a leaf, representing the clause, to its associated tree-node. For example, consider the CNF formula and its junction tree in Fig. 1. The shaded leaf nodes represent clauses.

The leaf-node factors corresponding to the CNF clauses have two roles. They materialize the evidence dictating that the corresponding clause is satisfied (any MPE assignment satisfies the formula). Second, they return a satisfying assignment to its variables with the maximum weight. We provide the factor's formal definition in the following section. Given an encoding of the PGM into a Boolean formula in CNF, the MPE assignment and probability can be read off from the model with the maximum weight.

Let $T_r(\mathcal{I})$ be a CNF-tree rooted at node r. For each tree-node index $i \in \mathcal{I}$, we denote by $F_i(\mathbf{X}_i)$ the factor associated with this node, T_i the subtree rooted at node i, and by f_i the subformula induced by the clauses in this subtree. For example, the formulae represented by subtrees T_1, T_2 rooted at nodes $1, 2$ respectively in Fig. 1 are $f_1 = \phi_1\phi_4\phi_6$ and $f_2 = \phi_2\phi_5\phi_7$. The children of tree-node i are denoted ch_i.

In the general setting, a separator set $\mathbf{S}_{i,j} = \mathbf{X}_i \cap \mathbf{X}_j$, between junction tree nodes i, j, enables inducing independence between variables on different sides of the edge only when *all* of the variables in $\mathbf{S}_{i,j}$ have been assigned a value (Darwiche 2009). We observe, however, that in CNF-trees this requirement may be too strict. Considering the CNF and corresponding CNF-tree in Figure 1, the *partial* assignment $\gamma = \{x_1\}$ renders the reduced formulas $f_1|\gamma = \phi_6$, $f_2|\gamma = \phi_5\phi_7$ and $f_3|\gamma = \emptyset$ disjoint even though variables X_2 and X_4 remained unassigned.

Discovering partial, rather than complete assignments, which decompose the formula will reduce factor sizes and enable more efficient inference. Therefore, a central notion in message-passing algorithms over CNF-trees is that of a *valid assignment* defined below.

Definition 1. *Let $T_r(\mathcal{I})$ be a CNF-tree rooted at node r and $i \in \mathcal{I}$ a node in T_r with $\mathrm{ch}_i = \{i_1, i_2, ..., i_m\}$. Let $\boldsymbol{Y} = \boldsymbol{y}$ be a partial assignment to $\boldsymbol{X}_i$. $\boldsymbol{y}$ is called a* valid (partial) assignment *to $\boldsymbol{X}_i$ if the following two conditions are satisfied:*

1. *$\boldsymbol{Y} = \boldsymbol{y}$ is consistent ($f_i|\boldsymbol{y} \neq 0$)*

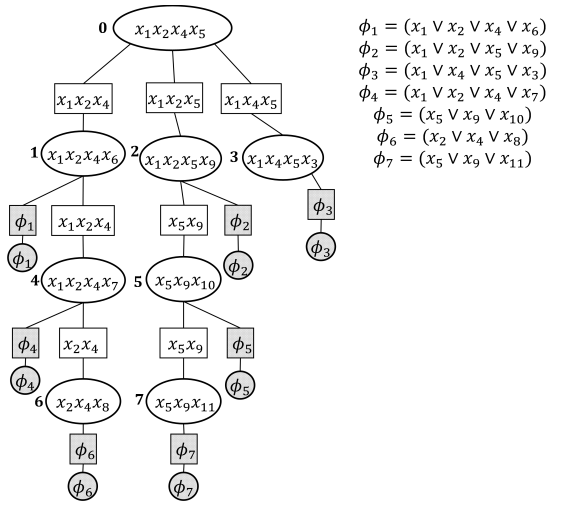
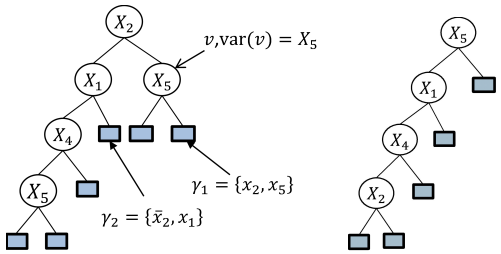


Figure 1: A CNF-tree for the formula $f = \bigwedge_{i=1}^{7} \phi_i$.



(a) F_0 (b) F_0'

Figure 2: Two possible tree-CPTs for root-node $i = 0$ ($\mathbf{X}_0 = \{X_1, X_2, X_4, X_5\}$) of the CNF-tree in Fig. 1.

2. *$f_i|\boldsymbol{y}$ is decomposed to pairwise-disjoint sub-formulas $f_1|\boldsymbol{y}, f_2|\boldsymbol{y}, ..., f_m|\boldsymbol{y}$.*

The factors of internal CNF-tree nodes are represented by tree-CPTs (Boutilier et al. 1996) where non-terminal vertices represent variables and terminal vertices correspond to the assignment defined by the path from the root.

Example 1. *An example of two possible tree-CPTs for the root-node $i = 0$ in the CNF-tree of Fig. 1 appear in Fig. 2. Note the terminal vertex in Fig. 2a that represents assignment $\gamma_2 = \{\bar{x}_2, x_1\}$. The partial assignment γ_2 induces subformulas $f_1|\gamma_2 = x_4 \vee x_8$, $f_2|\gamma_2 = \phi_5\phi_7$ and $f_3|\gamma_2 = \emptyset$, which correspond to subtrees T_1, T_2, T_3, respectively. Given assignment γ_2, these subformulas are consistent and pairwise disjoint, e.g., $\mathrm{var}(f_1|\gamma_2) \cap \mathrm{var}(f_2|\gamma_2) = \emptyset$, thus the partial assignment γ_2 is valid.*

From a performance point of view, our goal is to generate the smallest factor for each CNF-tree node. Namely, per each node, we would like to identify the smallest set of mutual exclusive and exhaustive *valid* partial assignments. The work in (Kenig and Gal 2015) provides techniques for reducing the size of tree-CPTs according to the CNF-tree topology and identifies a family of formulas with a linear-sized tree-CPTs whose MPE (and WMC) can be computed in P-time.

The runtime complexity of a message passing algorithm operating over a CNF-tree is exponential in a measure termed *CNF-tw*. Intuitively, this measure reflects the number of *distinct* CNFs resulting from the different valid assignments associated with a tree-CPT factor.

Basic MPE procedure with CNF-trees

The MPE assignment and weight for a Boolean formula in CNF are formally defined next.

Definition 2. *Let $f(\boldsymbol{X})$ denote a Boolean formula in CNF, $\boldsymbol{e}$ denote evidence, and $\boldsymbol{Q} = \boldsymbol{X} \setminus \text{var}(\boldsymbol{e})$ the set of unassigned variables. The MPE probability, given evidence $\boldsymbol{e}$, is defined as:*

$$Pr_{MPE}(f, \boldsymbol{e}) = \max_{\boldsymbol{q}: f|\boldsymbol{eq}=1} w(\boldsymbol{q})$$

The set of MPE assignments *that attain this maximal weight are defined as:*

$$MPE(f, \boldsymbol{e}) = \arg\max_{\boldsymbol{q}: f|\boldsymbol{eq}=1} w(\boldsymbol{q})$$

The leaf-factors (Def. 3), representing clauses, return the partial assignment with the maximal weight, satisfying the clause.

Definition 3. *Let i denote an incidence of a leaf node in a CNF-tree, representing the clause ϕ_i, and let $\boldsymbol{E} = \boldsymbol{e}$ denote some evidence. Let $\boldsymbol{U}_i = \text{var}(\phi_i) \setminus \boldsymbol{E}$ denote the unassigned variables in ϕ_i, $\boldsymbol{u}_i^* = \arg\max_{\boldsymbol{u}_i} w(\boldsymbol{u}_i)$ an assignment that maximizes the weights to its members, and $w(\boldsymbol{u}_i^*)$ its weight. Finally, let $v = \arg\max_{l \in \phi_i | \boldsymbol{e}} \left[\frac{\mathrm{w}_l}{\hat{\mathrm{w}}_l}\right]$ denote the literal with the largest positive to negative assignment ratio, and $\boldsymbol{u}_i^{**}$ the assignment in which literal v is set, and the rest of the literals are assigned as in $\boldsymbol{u}_i^*$. Then $F_{\phi_i} : \boldsymbol{e} \to \langle \mathbb{R}, \boldsymbol{u}_i \rangle$ is:*

$$F_{\phi_i}(\boldsymbol{e}) = \begin{cases} \langle 0, \emptyset \rangle & \text{if } \phi_i|\boldsymbol{e} = 0 \\ \langle w(\boldsymbol{u}_i^*), \boldsymbol{u}_i^* \rangle & \text{if } \phi_i|\boldsymbol{e}\boldsymbol{u}_i^* = 1 \\ \left\langle w(\boldsymbol{u}_i^*) \cdot \frac{\mathrm{w}_v}{\hat{\mathrm{w}}_v}, \boldsymbol{u}_i^{**} \right\rangle & \text{otherwise} \end{cases} \quad (1)$$

Example 2. *Consider clause $\phi_i = (x_1, \bar{x}_2, x_3)$. Assume without loss of generality that $\hat{\mathrm{w}}_x > \mathrm{w}_x$ for $x \in \{x_1, x_2, x_3\}$. Let us calculate $F_{\phi_i}(\boldsymbol{e})$ for $\boldsymbol{e}_1 = \bar{x}_2$ and $\boldsymbol{e}_2 = x_2$. Since $\phi_i|\boldsymbol{e}_1$ is satisfied then the factor will assign x_1 and x_3 with values that maximize their joint weight, i.e., $\boldsymbol{u}_i^* = \{\bar{x}_1, \bar{x}_3\}$, and the weight returned is $w(\boldsymbol{u}_i^*)$.*

This is not the case for $\boldsymbol{e}_2 = x_2$, which does not satisfy the clause. Moreover, the assignment to the unassigned literals that maximizes their joint weight, $\boldsymbol{u}_i^* = \{\bar{x}_1, \bar{x}_3\}$, does not satisfy the clause either. Therefore, we flip the value of the literal that will lead to the smallest decrease in the maximal weight. Assuming (w.l.o.g) that $x_1 = \arg\max_{l \in \phi_i | \boldsymbol{e}_2} \frac{\mathrm{w}_l}{\hat{\mathrm{w}}_l}$, then the resulting assignment returned by the factor is $\boldsymbol{u}_i^{**} = \{x_1, \bar{x}_3\}$.

It is worth noting that maximizing the partial assignment, returned by a leaf factor, can be computed in time that is linear in the number of variables in the clause.

Algorithm 1 presents the basic procedure for computing the MPE. It receives a rooted CNF-tree T_r, and evidence $\boldsymbol{e}$,

Algorithm 1: MPE_CNF_T($T_r, \boldsymbol{e}$), **returns** $\langle \Pr_{MPE}(f_r, \boldsymbol{e}), \text{MPE}(f_r, \boldsymbol{e}) \rangle$

```
1: if r is a leaf-node then
2:     return F_r(e) {By eq. 1}
3: end if
4: if cache(var(e), f_r|e) ≠ nil then
5:     ⟨Pr_MPE(f_r, e), MPE(f_r, e)⟩ ← cache(var(e), f_r|e)
6:     return ⟨Pr_MPE(f_r, e), MPE(f_r, e)⟩
7: end if
8: V_r^e ← var(f_r|e) {reduced formula vars}
9: MPE(f_r, e) ← 0.0 {init the return value}
10: while γ ← nextValid(γ, f_r|e, X_r \ var(e)) ≠ nil do
11:     V_r^{eγ} ← var(f_r|eγ) {reduced formula vars}
12:     β ← ∅ {The maximizing assignment to
        subsumed vars }
13:     for X ∈ V_r^e \ V_r^{eγ} do
14:         β ← β ∪ arg max_{x,x̄} (w_x, ŵ_x)
15:     end for
16:     γ ← γβ {Update the valid assignment}
17:     Pr(γ) ← w(γ) {Init assignment prob by its weight}
18:     for node n ∈ ch_r do
19:         ⟨Pr_MPE(f_n, eγ), MPE(f_n, eγ)⟩ ←
            MPE_CNF_T(T_n, eγ) {recurse}
20:         Pr(γ) ← Pr(γ) × Pr_MPE(f_n, eγ)
21:         γ ← γMPE(f_n, eγ) {update assignment according
            to the result from child}
22:     end for
23:     if Pr(γ) > Pr_MPE(f_r, e) then
24:         Pr_MPE(f_r, e) ← Pr(γ)
25:         MPE(f_r, e) ← γ
26:     end if
27: end while
28: cache(var(e), f_r|e) ← ⟨Pr_MPE(f_r, e), MPE(f_r, e)⟩
29: return ⟨Pr_MPE(f_r, e), MPE(f_r, e)⟩
```

and returns the MPE assignment and probability (Def. 2). If r is a leaf node, then the appropriate assignment and probability are returned in line 2 according to the leaf factor (Def. 3). If the cache contains the result for $f_r|\boldsymbol{e}$, it is returned in line 5. In line 8, the set of affecting variables, those that appear in at least one unsubsumed clause in $f_r|\boldsymbol{e}$, is placed in $\mathbf{V}_r^{\boldsymbol{e}}$.

Valid assignments are processed in lines 10-27. Generating the set of valid assignments is described in (Kenig and Gal 2015). Each valid assignment, γ, induces the reduced subformula $f_r|\boldsymbol{e}\gamma$. Since the newly subsumed variables ($\mathbf{V}_r^{\boldsymbol{e}} \setminus \mathbf{V}_r^{\boldsymbol{e}\gamma}$) do not affect the formula, they can be assigned the value maximizing their weight (lines 13-15).

The children of r are processed in lines 18-22. The algorithm is recursively called on each of r's subtrees, T_n, in line 19. After its return, the MPE probability and assignment, given evidence $\boldsymbol{e}\gamma$, are updated in lines 20 and 21, respectively. By definition of valid assignments (Def. 1), the subformulas corresponding to r's children are disjoint, enabling the update.

$\text{MPE}(f_r, \boldsymbol{e})$, the model of $f_r|\boldsymbol{e}$ with the highest probability, is set in lines 23-26. Since the valid assignments are mutual exclusive and exhaustive, the maximization operator, effectively applied in lines 23-26, returns the expected value.

Finally, the result is cached and returned in lines 28 and 29, respectively.

Bounding and pruning

We can improve the performance of algorithm MPE_CNF_T by pruning a portion of the search space, at each CNF-tree-node, using an upper bound on the MPE probability. Suppose for example that the algorithm is called with CNF-tree-node r, and evidence $\boldsymbol{e}$. If we have already encountered a complete assignment that has probability p, and we know that every completion of instantiation $\boldsymbol{e}$ (to all variables in f_r, $\text{var}(f_r)$) will not lead to a higher probability, we can abandon this partial assignment as it would not lead to instantiations that improve on the one previously found.

Let $\Pr^u_{MPE}(f_r, \boldsymbol{e})$ stand for the calculated upper bound of the MPE probability $\Pr_{MPE}(f_r, \boldsymbol{e})$ (updated in line 24). We can modify Algorithm MPE_CNF_T by inserting the following line just before line 8:

$$\textbf{if } \Pr^u_{MPE}(f_r, \boldsymbol{e}) \leq p, \textbf{return}$$

The efficiency of the resulting algorithm will depend on the quality of the calculated upper bounds and the time it takes to compute them. Furthermore, our goal is to compute the tightest bounds possible under certain time and memory constraints.

The bound, p, can be applied in other stages of the algorithm as well. For example, assume that during the generation of the tree-CPT in procedure nextValid (line 10), a certain tree-CPT branch induces a probability that falls below p. Obviously, any extension of this assignment to one that is valid will induce an MPE probability below p, enabling us to prune this partial assignment, and proceed in searching for a valid MPE assignment in a different branch of the tree-CPT.

Now, let us assume that the weight of a valid assignment, γ, denoted $w(\gamma)$ is greater than p. In the case that during the processing of r's children in lines 18-22, the assignment returned from one of the children causes the computed probability to fall below p, we can abandon the valid assignment γ, *without* processing the rest of the children, thereby pruning a large portion of the search space.

All of these pruning techniques are applied recursively to the children of the CNF-tree node by adjusting the known MPE probability, p, according to the current evidence $\boldsymbol{e}$, valid assignment γ and the weights returned from the other children.

Generating upper bounds

A technique that generates an array of upper bounds would allow us to exchange the bound tightness with the time it takes to compute it. Both the mini-buckets (Dechter and Rish 1998), and the *node-splitting* (Choi, Chavira, and Darwiche 2007; Choi, Standley, and Darwiche 2009; Choi and Darwiche 2010) techniques operate in this fashion. We provide an intuition of our approach using a simplified case. Let $f(\mathbf{X})$ denote a Boolean function in CNF such that

$$f(\mathbf{X}) = f_1(\mathbf{X}_1, Y) \cdot f_2(\mathbf{X}_2, Y)$$

$Y \in \mathbf{X}$ and $\mathbf{X}_1 \cap \mathbf{X}_2 = \emptyset$. Furthermore, assume that $\text{w}_y = \hat{\text{w}}_{\bar{y}} = 0.5$, then

$$\begin{aligned}
\Pr_{MPE}(f(\mathbf{X})) &= \max_{\mathbf{x}} \Pr(f(\mathbf{x})) \\
&= \max_{y} \left[\max_{\mathbf{x}_1} \Pr(f_1(\mathbf{x}_1, y)) \cdot \max_{\mathbf{x}_2} \Pr(f_2(\mathbf{x}_2, y)) \right] \\
&= \beta \left[\max_{\mathbf{x}_1, y_1} \Pr(f_1(\mathbf{x}_1, y_1)) \right] \left[\max_{\mathbf{x}_2, y_2} \Pr(f_2(\mathbf{x}_2, y_2)) \right] \texttt{Eq}(y_1, y_2) \\
&\leq \beta \left[\max_{\mathbf{x}_1, y_1} \Pr(f_1(\mathbf{x}_1, y_1)) \right] \left[\max_{\mathbf{x}_2, y_2} \Pr(f_2(\mathbf{x}_2, y_2)) \right]
\end{aligned} \tag{2}$$

where Eq is the equality predicate, and $\beta = 2$.

When f is an encoding of a Bayesian network to a knowledge-base in CNF, and Y is an *indicator* variable (i.e., has a specific role in the translation to CNF and $\text{w}_y = \hat{\text{w}}_{\bar{y}} = 1.0$, see (Chavira and Darwiche 2008)), then $\beta = 1.0$. This is actually an instance of *variable-splitting* where Y is split into two variables Y_1, Y_2 (Choi, Chavira, and Darwiche 2007). Once the equality constraint is relaxed, the formula becomes less constrained enabling the generation of a higher probability. In other words, relaxing the formula by ignoring certain dependency constraints between its variables results in upper bounds that can be used to prune the search space in the MPE_CNF_T algorithm. Clearly, the accuracy of the bound deteriorates as the size of the ignored dependencies, or split variables, increases.

We quantify the complexity of computing upper bounds for CNF-tree nodes using the *CNF-tw* parameter (Kenig and Gal 2015). That is, the computational effort put into computing these bounds should be no greater than exactly solving a formula whose *CNF-tw* $= k$. If the length of each tree-CPT branch, generated during the algorithm, were limited to k, then, in the worst case, the tree-CPT would grow into a full binary tree whose height is $k + 1$. Since the number of leaves in such a tree is exponential in its height, i.e., 2^{k+1}, then the number of leaves, representing assignments, would be limited to 2^{k+1}, restricting the complexity as required.

We refer to this restriction as a k-bound. Executing algorithm MPE_CNF_T with this bound causes it to generate tree-CPTs whose assignments contain at most k decision variables. Since these assignments do not necessarily decompose the formula i.e., they are not *valid* assignments (Def. 1), they result in upper bounds to the actual probabilities. These upper bounds are stored in a distinct cache and are then used to prune the search space. Broadly, the procedure is as follows.

1. Execute MPE_CNF_T with k-bound=k. This guarantees that each valid assignment processed by the algorithm will contain at most k assigned decision variables.
2. Save the cache contents as an Upper-Bound-Cache that will hold the upper bound approximations for the corresponding formulas.
3. Generate a new, empty cache object.
4. Execute MPE_CNF_T with no k-bound and the new cache object while extracting the required upper bounds from Upper-Bound-Cache.

Determining the optimal bound, k, per instance is deferred as part of future research.

Related Work

Current approaches for answering the MPE query compute upper bounds by applying an exact inference algorithm on an approximate model. *Mini-buckets* is an approximate inference scheme, employed in branch and bound algorithms for computing MPE (Dechter and Rish 1998). Using this approach, the variable elimination algorithm is applied to the problem as long as computational resources allow it. When this is not the case, the mini-buckets algorithm will heuristically ignore certain dependencies, enabling the variable elimination algorithm to proceed. The mini-buckets approach can therefore be viewed as an algorithm which calculates the bound by applying an exact inference algorithm on an approximate model, generated by removing dependencies from the original one (Choi, Chavira, and Darwiche 2007).

A different approach, called *Node-Splitting* (Choi, Chavira, and Darwiche 2007; Choi and Darwiche 2010) computes the bounds by solving MPE queries on a relaxed, or simplified network. The relaxation is achieved by *splitting* nodes of the network such that its treewidth decreases, making it tractable to an exact inference algorithm. Since the result on the relaxed model is no less than that of the original, the resulting figures may be used as upper bounds in a branch and bound search algorithm for MPE.

`Clone`[1] (Pipatsrisawat and Darwiche 2007) is a MAX-SAT solver that applies node splitting, knowledge compilation, and search. There is a simple reduction between MPE and MAX-SAT and methods for applying a MAX-SAT algorithm for solving the MPE problem have been introduced (Park 2002). `Clone` applies node-splitting to bring the network to a tractable form, namely, nodes are split until the network's width reaches the required limit. The split network is then compiled, using the `C2D` compiler to a d-DNNF circuit. Finally, the compiled circuit is used to calculate upper bounds of various partial assignments during the branch and bound search algorithm.

In all approaches for upper-bounding the probability, including mini-buckets and node-splitting, there is a trade-off between the tightness of the bounds of the applied heuristic, and the efficiency of computing them. In that regard, our approach is no different. A larger k-bound results in a lower number of variables shared between formulas solved independently, tightening the bound. A lower k-bound, on the other hand, results in lower computational complexity. Our approach combines this property with the ability to utilize the junction-tree structure for speeding up the optimization algorithm.

Preliminary Experimental Results

We compare our solver to `DAOOPT`[2], an AND/OR branch-and-bound solver for combinatorial optimization problems expressed as MPE queries over PGMs (Marinescu and Dechter 2009), and to `Clone` (Pipatsrisawat and Darwiche 2007), a MAX-SAT solver that applies node splitting, knowledge compilation, and search.

[1] Available at: http://reasoning.cs.ucla.edu/clone/

[2] Available online at: https://github.com/lotten/daoopt

We evaluate the proposed approach on a set of benchmark networks from the UAI probabilistic inference challenge. The `DAOOPT` solver contains a large range of settings, therefore we opted for the default configuration. We also compare our solver with `Clone` (Pipatsrisawat and Darwiche 2007) on a set of randomly generated networks. We implemented our algorithm in `C++`[3] and carried out the experiments on a 2.33GHz quad-core `AMD64` with 8GB of RAM running `CentOS` Linux 6.6. Individual runs were limited to a 2000-second time-out. In what follows, `MPE_CNF_T` refers to our implementation of Algorithm 1 with the bounding and pruning techniques.

Grid networks The experiment results for the Grid networks are available in Table 1. These experiments were executed with a k-bound$= 3$. On the Grid benchmark with 50% deterministic CPTs, `DAOOPT` outperformed `MPE_CNF_T` on 49 out of the 55 instances, completed by both solvers. For the networks exhibiting a larger percentage of deterministic factors, the result is reversed. On the networks with a 75% deterministic ratio, `MPE_CNF_T` completed 82 of the 110 instances while `DAOOPT` managed to complete only 54 instances within the designated timeout. On the 90% deterministic ratio `MPE_CNF_T` completed 92 of the 100 instances, compared with 38 completed by `DAOOPT`.

Observing the relationships between the *CNF-tw* (Kenig and Gal 2015) and induced-width (Marinescu and Dechter 2009) of these networks explains these results. On the instances exhibiting a large degree of local structure, the *CNF-tw* is substantially smaller than the induced width, leading to better performance of the `MPE_CNF_T` solver. This can be attributed to the ability of `MPE_CNF_T` to exploit this local structure by use of unit propagation and clause learning.

For the 50% grid networks, the *CNF-tw* is only slightly lower than the induced width indicating that the additional computational effort required for performing inference over the CNF is not worthwhile.

Net	`MPE_CNF_T` **Avg./Med. Runtime**	`DAOOPT` **Avg./Med. Runtime**	**Avg./Med. *CNF-tw***	**Avg./Med. induced-Width**
Grid 50	232/91	115/4	19.9/20.2	21.1/21
Grid 75	99/31	335/52	18.4/18	26.4/26.5
Grid 90	2/1.5	419/84.5	11.7/11.8	32.4/32

Table 1: Grid networks results. Runtime values are in seconds.

Promedas The instances of this benchmark were executed with a k-bound of 3. Out of the 238 networks, `MPE_CNF_T` was able to process 126 networks within the designated timeout of 2000 seconds, while `DAOOPT` completed 105. `MPE_CNF_T` outperformed `DAOOPT` on 68 instances, while `DAOOPT` outperformed `MPE_CNF_T` on 34. The results are plotted in Fig. 3.

Randomly Generated Networks We tested the `MPE_CNF_T` solver on a set of randomly generated

[3] Code is available at: https://github.com/batyak/PROSaiCO/

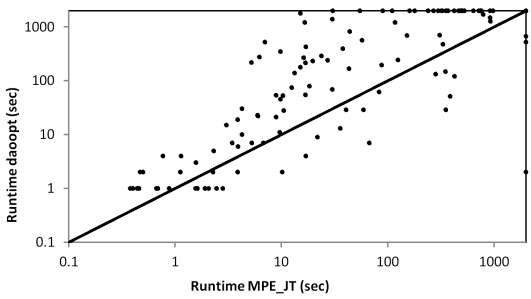


Figure 3: Promedas benchmark: Scatter plot of the runtime in seconds for each problem instance using `MPE_CNF_T` (x-axis) and `DAOOPT` (y-axis). Points above the $y = x$ line represent problem instances where `MPE_CNF_T` is better. Axes are log-scale.

Bayesian networks and compared the performance to the `Clone` solver[4]. We generated a total of 132 networks with varying parameters of size, domain, connectivity and percentage of deterministic factors. Full details will be made available in the complete version of this paper.

On this set of benchmarks `MPE_CNF_T` vastly outperformed `Clone` as can be seen in Table 2. Our solver's advantage stems from the fact that it exploits the structural properties of the CNF-tree, while `Clone` was able to utilize only local structure encoded into the CNF.

	`MPE_CNF_T`	`Clone`
# Processed	132	15
Average Runtime	28.1	290
Median Runtime	1.83	125
Average *CNF-tw*	9.5	–
Median *CNF-tw*	9.5	–

Table 2: Summary of results over randomly generated Bayesian networks. Runtime values are in seconds and the timeout is 2000 seconds.

References

Boutilier, C.; Friedman, N.; Goldszmidt, M.; and Koller, D. 1996. Context-specific independence in bayesian networks. In *UAI*, 115–123.

Capelli, F.; Durand, A.; and Mengel, S. 2014. Hypergraph acyclicity and propositional model counting. In *SAT*, 399–414.

Chavira, M., and Darwiche, A. 2008. On probabilistic inference by weighted model counting. *Artif. Intell.* 172(6-7):772–799.

Choi, A., and Darwiche, A. 2010. Relax, compensate and then recover. In *New Frontiers in Artificial Intelligence - JSAI-isAI 2010 Workshops, LENLS, JURISIN, AMBN, ISS, Tokyo, Japan, November 18-19, 2010, Revised Selected Papers*, 167–180.

[4]Available online at http://reasoning.cs.ucla.edu/clone/

Choi, A.; Chavira, M.; and Darwiche, A. 2007. Node splitting: A scheme for generating upper bounds in bayesian networks. In *UAI 2007, Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, Vancouver, BC, Canada, July 19-22, 2007*, 57–66.

Choi, A.; Standley, T.; and Darwiche, A. 2009. Approximating weighted max-sat problems by compensating for relaxations. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, 211–225.

Darwiche, P. A. 2009. *Modeling and Reasoning with Bayesian Networks*. New York, NY, USA: Cambridge University Press, 1st edition.

Dechter, R., and Rish, I. 1998. Mini-buckets: A general scheme for approximating inference. *Journal of ACM* 50:2003.

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

Jensen, F. V.; Lauritzen, S. L.; and Olesen, K. G. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly* 4:269–282.

Kenig, B., and Gal, A. 2015. On the impact of junction-tree topology on weighted model counting. In *Scalable Uncertainty Management - 9th International Conference, SUM 2015, Québec City, QC, Canada, September 16-18, 2015. Proceedings*, 83–98.

Kenig, B.; Gal, A.; and Strichman, O. 2013. A new class of lineage expressions over probabilistic databases computable in p-time. In *SUM*, 219–232.

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

Marinescu, R., and Dechter, R. 2009. AND/OR branch-and-bound search for combinatorial optimization in graphical models. *Artif. Intell.* 173(16-17):1457–1491.

Park, J. D. 2002. Using weighted max-sat engines to solve mpe. In *Eighteenth National Conference on Artificial Intelligence*, 682–687. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Pipatsrisawat, K., and Darwiche, A. 2007. Clone: Solving weighted max-sat in a reduced search space. In *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007, Proceedings*, 223–233.

Szeider, S. 2011. The parameterized complexity of constraint satisfaction and reasoning. In *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, 27–37.

Zhang, N. L., and Poole, D. 1996. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research* 5:301–328.