Anytime Recursive Best-First Search for Bounding Marginal MAP

Radu Marinescu,[1] Akihiro Kishimoto,[1] Adi Botea,[1] Rina Dechter,[2] Alexander Ihler[2]
[1]IBM Research
[2]University of California, Irvine
radu.marinescu@ie.ibm.com,akihirok@ie.ibm.com,adibotea@ie.ibm.com, dechter@ics.uci.edu,ihler@ics.uci.edu

Abstract

Marginal MAP is a difficult mixed inference task for graphical models. Existing state-of-the-art solvers for this task are based on a hybrid best-first and depth-first search scheme that allows them to compute upper and lower bounds on the optimal solution value in an anytime fashion. These methods however are memory intensive schemes (via the best-first component) and do not have an efficient memory management mechanism. For this reason, they are often less effective in practice, especially on difficult problem instances with very large search spaces. In this paper, we introduce a new recursive best-first search based bounding scheme that operates efficiently within limited memory and computes anytime upper and lower bounds that improve over time. An empirical evaluation demonstrates the effectiveness of our proposed approach against current solvers.

Introduction

Graphical models provide a powerful framework for reasoning about conditional dependency structures over many variables. The Marginal MAP (MMAP) query asks for the optimal configuration of a subset of variables that has the highest marginal probability. Specifically, MMAP distinguishes between maximization variables (called MAP variables) and summation variables, and it is more difficult than either max- or sum- inference tasks alone primarily because summation and maximization operations do not commute, forcing processing along constrained variable orderings that may have significantly higher induced widths (Dechter 1999; 2013). This implies larger search spaces (when using search algorithms) or larger messages (when using message-passing schemes). In general, MMAP is NP^{PP}-complete and it can be NP-hard even on tree structured models (Park 2002). Still, MMAP is often the appropriate task where hidden variables or uncertain parameters exist. It can be also treated as a special case of the more complicated frameworks of decision networks (Howard and Matheson 2005; Liu and Ihler 2013).

Over the past few years, a prominent direction of research has focused on a new generation of solvers that provide not only anytime lower bounds but also anytime upper bounds on the optimal MMAP value. We distinguish two classes of solvers that can provide either deterministic or probabilistic bounds. Indeed, the most advanced deterministic bounding schemes are based on a hybrid best+depth-first search strategy over an AND/OR search graph. The best-first component aims at generating improved upper bounds while the depth-first search facilitates the generation of improved lower bounds, which are obtained by solving exactly the corresponding conditioned summation subproblems. For this reason, these methods are limited to problems with tractable summation subproblems (Marinescu et al. 2017).

On the other hand, the probabilistic bounding algorithms were introduced recently to address the case when the conditioned summation subproblem is intractable (Marinescu, Dechter, and Ihler 2018). Indeed, these methods, which in general do not come with completeness guarantees, combine stochastic best-first search strategies with concentration bounds on the conditioned likelihood computations in order to produce probabilistic upper and lower bounds with high confidence in an anytime fashion.

A major drawback of these recent deterministic and probabilistic solvers is that they are essentially memory intensive schemes (via the best-first component) and do not use an efficient memory management mechanism. When the available memory is exhausted, improving the upper bound via the best-first search component is no longer possible. In this case, the only option is to switch to depth-first search and try to improve the current lower bound for the remaining time. Therefore, these solvers are often less effective in practice, especially under more restrictive memory requirements.

Contribution In this paper, we focus on deterministic bounding schemes and introduce a new memory efficient anytime scheme for computing deterministic upper and lower bounds on the optimal MMAP value. In addition, our proposed approach is also complete, namely it guarantees finding the optimal solution if given enough time. More specifically, we develop an anytime recursive best-first search algorithm that traverses the AND/OR search space for MMAP and uses a bounded size cache table to store partial search results that allows it to operate efficiently within restricted memory. Therefore, our proposed algorithm replaces the best-first search component (which is used by the existing state-of-the-art hybrid best+depth-first schemes) with recursive best-first search that yields best-first behavior with any memory bound.

However, unlike regular recursive best-first search which backs up the value of a node only to its parent, our algorithm propagates the node values all the way up to the root of the search space. The important benefit of this scheme is that it can generate upper bounds in an anytime and anyspace manner. Furthermore, unlike the previous hybrid schemes, our approach applies search over the entire AND/OR graph, including the conditioned summation problems. The earlier algorithms search the MAP subspace only and exactly solve the summation subproblems by whatever means. The inclusion of the summation subspace can facilitate more gradual upper bounds that improve with time, as we will show in our experiments. Notice that for pure optimization (maximization), a lower bound is obtained by the best current solutions that are found. In contrast, for MMAP, computing the cost of a MAP solution exactly may not be possible at all (current state-of-the-art schemes must solve the summation subproblems exactly). Therefore, our new algorithm explores the conditioned sum spaces partially, and computes heuristic lower bounds on the tip nodes, which are subsequently propagated back to generate a global lower bound. For efficiency, these local lower bounds can be precomputed using any lower bounding scheme for conditioned likelihood, but in our work we illustrate this methodology using the weighted mini-bucket approximation (Liu and Ihler 2011).

Our empirical evaluation on various difficult benchmarks demonstrates the effectiveness of the new bounding scheme compared with the most recent deterministic and probabilistic bounding approaches.

Background

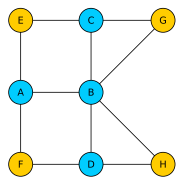
A *graphical model* is a tuple $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_i : i \in V\}$ is a set of variables indexed by set V and $\mathbf{D} = \{D_i : i \in V\}$ is the set of their finite domains of values. $\mathbf{F} = \{\psi_\alpha(\mathbf{X}_\alpha) : \alpha \in F\}$ is a set of discrete non-negative real-valued factors indexed by set F, where each ψ_α is defined on a subset of variables $\mathbf{X}_\alpha \subseteq \mathbf{X}$, called its scope. Specifically, $\psi_\alpha : \Omega_\alpha \to \mathbb{R}_+$, where Ω_α is the Cartesian product of the domains of each variable in $\mathbf{X}_\alpha$. The scopes of the factors define a *primal graph* whose vertices are the variables and whose edges connect any two variables that appear in the scope of the same factor. The model $\mathcal{M}$ defines a factorized probability distribution on $\mathbf{X}$,

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{\alpha \in F} \psi_\alpha(\mathbf{x}_\alpha)$$

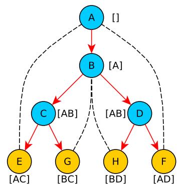
The *partition function*, Z, normalizes the probability.

Let $\mathbf{X}_M = \{X_1, ..., X_m\}$ be a subset of $\mathbf{X}$ called MAP variables and $\mathbf{X}_S = \mathbf{X} \setminus \mathbf{X}_M$ be the complement of $\mathbf{X}_M$, called sum variables. The Marginal MAP (MMAP) task seeks an assignment $\mathbf{x}_M^*$ to variables $\mathbf{X}_M$ having maximum probability. This requires access to the marginal distribution over $\mathbf{X}_M$, which is obtained by summing out variables $\mathbf{X}_S$:

$$\mathbf{x}_M^* = \underset{\mathbf{X}_M}{\operatorname{argmax}} \sum_{\mathbf{X}_S} \prod_{\alpha \in F} \psi_\alpha(\mathbf{x}_\alpha) \qquad (1)$$



(a) Primal graph



(b) Pseudo tree

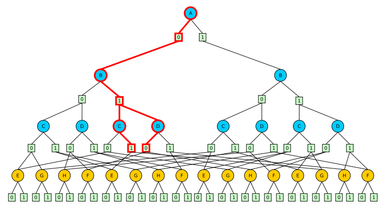

(c) AND/OR search graph

Figure 1: A simple graphical model.

AND/OR Search Spaces

A significant recent improvement in search for MMAP inference has been achieved by using AND/OR search spaces, which often capture problem structure far better than standard OR search methods (Marinescu, Dechter, and Ihler 2014; Dechter and Mateescu 2007). The AND/OR search space is defined relative to a *pseudo tree* of the primal graph, which captures problem decomposition.

Definition 1 (pseudo tree)**.** *A* pseudo tree *of an undirected graph $G = (V, E)$ is a directed rooted tree $\mathcal{T} = (V, E')$ such that every arc of G not in E' is a back-arc in $\mathcal{T}$ connecting a node in $\mathcal{T}$ to one of its ancestors. The arcs in E' may not all be included in E.*

For MMAP we need to restrict the collection of pseudo trees to *valid* ones only. Specifically, given a graphical model $\mathcal{M}$ with primal graph G, a pseudo tree $\mathcal{T}$ of G is *valid for the MAP variables* $\mathbf{X}_M$ if $\mathcal{T}$ restricted to $\mathbf{X}_M$ forms a connected *start pseudo tree* having the same root as $\mathcal{T}$.

Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ with primal graph G and valid pseudo tree $\mathcal{T}$ of G, the *AND/OR search tree* $S_\mathcal{T}$ based on $\mathcal{T}$ has alternating levels of OR nodes corresponding to the variables, and AND nodes corresponding to the values of the OR parent's variable, with edge weights extracted from the original functions $\mathbf{F}$ (for details see (Dechter and Mateescu 2007)). Identical sub-problems, identified by their *context* (the partial instantiation that separates the sub-problem from the rest of the problem graph), can be merged, yielding an *AND/OR search graph*. Merging all context-mergeable nodes yields the *context minimal AND/OR search graph*, denoted $C_\mathcal{T}$. The size of $C_\mathcal{T}$ is exponential in the induced width of G along a depth-first traversal of $\mathcal{T}$ (also known as the *constrained* induced width).

Definition 2 (solution subtree)**.** *A* solution subtree $\hat{x}_M$ *of $C_\mathcal{T}$ relative to the MAP variables $\mathbf{X}_M$ is a subtree of $C_\mathcal{T}$ restricted to $\mathbf{X}_M$ that: (1) contains the root of $C_\mathcal{T}$; (2) if an internal OR node $n \in C_\mathcal{T}$ is in $\hat{x}_M$, then n is labeled with a MAP variable and exactly one of its children is in $\hat{x}_M$; (3)*

if an internal AND node $n \in C_{\mathcal{T}}$ is in $\hat{x}_M$ then all its OR children which denote MAP variables are in $\hat{x}_M$.

Each node n in $C_{\mathcal{T}}$ can be associated with a *value* $v(n)$ capturing the optimal marginal MAP value of the conditioned sub-problem rooted at n, while for a sum variable it is the likelihood of the partial assignment denoted by n. Clearly, $v(n)$ can be computed recursively based on the values of n's successors: OR nodes by maximization or summation (for MAP or sum variables), and AND nodes by multiplication.

Example 1. *Figure 1 shows a graphical model with 8 bi-valued variables and 11 binary factors, where the MAP and sum variables are $\mathbf{X}_M = \{A, B, C, D\}$ and $\mathbf{X}_S = \{E, F, G, H\}$, respectively. Figure 1(a) is the primal graph while Figure 1(b) is a valid pseudo tree whose MAP variables form a start pseudo tree (dashed lines denote back-arcs). Figure 1(c) displays the context minimal AND/OR search graph based on the pseudo tree (the contexts are shown next to the pseudo tree nodes). A solution subtree corresponding to the MAP assignment (A=0,B=1,C=1,D=0) is shown in red.*

Deterministic Bounding Schemes

The most advanced deterministic anytime methods for bounding MMAP are based on hybrids of *best-first* and *depth-first* AND/OR search in order to facilitate the generation of upper bounds (via the best-first component) alongside lower bounds (via the depth-first component) in an anytime fashion (Marinescu et al. 2017). Specifically, *best-first AND/OR search with depth-first lookaheads* (LAOBF) traverses the search space in a best-first manner while performing explicit depth-first dives (or lookaheads) below the leaf nodes of the best partial solution tree. *Alternating best-first with depth-first AND/OR search* (AAOBF) is a parameter-free scheme that interleaves an outer best-first loop with an aggressive depth-first loop that aims to find improved suboptimal solutions as quickly as possible. *Learning depth-first AND/OR search* (LnDFS) is also a memory-intensive algorithm that consists of a sequence of depth-first search iterations to find improved feasible solutions that yield tighter lower bounds. Each depth-first search iteration is then followed by a bottom-up update of the heuristic node values to provide improved upper bounds. Recent evaluations on various difficult benchmarks demonstrated the effectiveness of these schemes compared with previous anytime solvers (Maua and Campos 2012; Lee et al. 2016; Marinescu et al. 2017).

Weighted Mini-Bucket Heuristics

The effectiveness of the above search algorithms greatly depends on the quality of the upper bound heuristic function that guides the search. Specifically, algorithms AAOBF, LAOBF and LnDFS use a weighted mini-bucket (WMB) based heuristic (Liu and Ihler 2011) which proved to be quite effective over the past years. First, WMB improves the naïve mini-bucket bound (Dechter and Rish 2003) with Hölder's inequality. For a given variable X_k, the mini-buckets Q_{kr} associated with X_k are assigned a non-negative *weight* $w_{kr} \geq 0$, such that $\sum_r w_{kr} = 1$. Then, each mini-bucket r is eliminated using a powered sum, $(\sum_{X_k} f(X)^{1/w_{kr}})^{w_{kr}}$. Subsequently, the cost shifting scheme (or reparameterization) is performed across mini-buckets to match the marginal beliefs (or "moments") to further tighten the bound. The single-pass message passing algorithm yields a scheme denoted by WMB$_u(i)$, where i is called the i-bound and controls the accuracy of the approximation (Dechter and Rish 2003). The heuristic can be pre-compiled along a reversed depth-first traversal of a pseudo tree and used to guide the search (Kask and Dechter 2001; Marinescu, Dechter, and Ihler 2014).

Algorithm 1: GENERATEANDPROPAGATE(n)

```
Input: node $n$
1  if $n$ labeled $\langle X_i \rangle$ is OR node then
2      forall values $x_i \in D_{X_i}$ do
3          create AND node $c_i$ labeled $\langle X_i, x_i \rangle$
4          let $ct$ = Context($c_i$)
5          if $ct$ is in the cache table then
6              $(q(c_i), l(c_i), c_i.solved)$ = ReadCache($ct$)
7          else $(q(c_i), l(c_i), c_i.solved) = (h(c_i), lb(c_i), \textbf{false})$
8  else if $n$ labeled $\langle X_i, x_i \rangle$ is AND node then
9      forall $X_i$'s children $X_j$ in $\mathcal{T}$ do
10         create OR node $c_i$ labeled $\langle X_j \rangle$
11         let $ct$ = Context($c_i$)
12         if $ct$ is in the cache table then
13             $(q(c_i), l(c_i), c_i.solved)$ = ReadCache($ct$)
14         else $(q(c_i), l(c_i), c_i.solved) = (h(c_i), lb(c_i), \textbf{false})$
15 forall nodes $m$ along the path from $n$ to root $s$ do
16     if $m$ is OR node labeled by MAP variable then
17         $q(m) = \max_{c_i \in ch(m)}(w_{(m,c_i)} \cdot q(c_i))$
18         $l(m) = w_{(m,c_j)} \cdot l(c_j)$, where $c_j$ is $m$'s child on the path
19     else if $m$ is OR node labeled by SUM variable then
20         $q(m) = \sum_{c_i \in ch(m)} w_{(m,c_i)} \cdot q(c_i)$
21         $l(m) = \sum_{c_i \in ch(m)} w_{(m,c_i)} \cdot l(c_i)$
22     else if $m$ is AND node then
23         $q(m) = \prod_{c_i \in ch(m)} q(c_i)$, $l(m) = \prod_{c_i \in ch(m)} l(c_i)$
24 let $\mathcal{U} = q(s)$ and $\mathcal{L} = \max(\mathcal{L}, l(s))$; output $\langle \mathcal{L}, \mathcal{U} \rangle$
```

Anytime Recursive Best-First AND/OR Search

In this section, we describe a new memory efficient best-first search scheme that computes anytime deterministic upper and lower bounds on the optimal MMAP value. Specifically, we develop RBFAOO+, a recursive best-first AND/OR search algorithm that operates with restricted memory and (1) uses a threshold controlling mechanism to explore the AND/OR search space in a depth-first like manner, (2) employs an overestimation technique to reduce the overhead of re-expanding internal nodes, and (3) uses a back propagation mechanism to facilitate the computation of improved upper and lower bounds in an anytime fashion. We should emphasize that although recursive best-first AND/OR search was previously used as an exact method for pure MAP inference (Kishimoto and Marinescu 2014) and MMAP (Marinescu, Dechter, and Ihler 2015), this is the first time it is extended into a complete anytime search scheme for bounding MMAP.

The basic idea behind our best-first search based approach is to associate each node in the search space with two values

Algorithm 2: BESTCHILD(n)

```
Input: node $n$ (OR node labeled by MAP variable)
1  $q = q_2 = \beta = -\infty$ and $n.solved$ = false
2  forall $n$'s child $c_i$ do
3      $ct$ = Context($c_i$)
4      if $ct \in Cache$ then $(q(c_i), s)$ = ReadCache($ct$)
5      else $(q(c_i), s) = (h(c_i), \text{false})$
6      $q_{c_i} = w_{(n,c_i)} \cdot q(c_i)$
7      if $s$ = true then $\beta = \max(\beta, q_{c_i})$
8      if $q_{c_i} > q$ or ($q_{c_i} = q$ and $n.solved$ = false) then
9          $q_2 = q$, $n.solved = s$, $q = q_{c_i}$ and $c_{best} = c_i$
10     else if $(q_{c_i} > q_2)$ then $q_2 = q_{c_i}$
11 return $(c_{best}, q, q_2, \beta)$
```

Algorithm 3: UNSOLVEDCHILD(n)

```
Input: node $n$
1  $q_{best} = -\infty$
2  forall $n$'s child $c_i$ do
3      $ct$ = Context($c_i$)
4      if $ct \in Cache$ then $(q(c_i), s)$ = ReadCache($ct$)
5      else $(q(c_i), s) = (h(c_i), \text{false})$
6      if $s$ = false and $q(c_i) > q_{best}$ then
7          $q_{best} = q(c_i)$ and $c_{best} = c_i$
8  return $(c_{best}, q_{best})$
```

denoted by $q(n)$ and $l(n)$, representing an upper bound and a lower bound on the value of the subproblem rooted at n, respectively. During search, these node values are revised recursively based on the values of their successors and then propagated bottom up to the root node in order to generate global upper and lower bounds that improve over time. It is important to note that, unlike the existing deterministic bounding schemes which conduct the hybrid best+first-search over the MAP variables only, our approach explores best-first the *entire* AND/OR search space, including the sub-space defined by the conditioned summation subproblem.

Notations

We use $\mathcal{U}$ and $\mathcal{L}$ to denote the current best global upper and lower bounds on the optimal MMAP value. For node n, $ch(n)$ denotes its children, while $w_{(n,m)}$ is the weight labeling the arc $n \rightarrow m$ in the search space. $Cache$ is the cache table that stores the partial search results. It is implemented as a hash table with the Zobrist function (Zobrist 1970) using 96-bit integers to compute the hash keys of the node contexts. When the cache table is filled up and new results need to be stored there, some cached results must be replaced. We use SmallTreeGC (Nagai 1999), a batch-based replacement that discards $R\%$ of the table entries with small subtree sizes. Functions `ReadCache` and `WriteCache` are used to read and write the cache table entries. Algorithm 1 describes the node generation and node values propagation during search, while Algorithms 2 and 3 show how the best child node and, respectively, an unsolved child node is selected for expansion.

Algorithm 4: RBFAOO+

```
Input: node $n$
1  Function RBFAOO+($root$):
2      $root.th = 0$
3      $q$ = RBFS($root$)
4      return $q$
5  Function RBFS($n$):
6      if $ch(n) = \emptyset$ then
7          SaveCache(Context(n), $q = 1$, $l = 1$, $s = true$)
8          return $q$
9      GenerateAndPropagate($n$)
10     if $n$ is an OR node then
11         while true do
12             if $n$ is labeled by MAP variable then
13                 $(c_{best}, q, q_2, \beta)$ = BestChild($n$)
14                 if $q < n.th$ or $n.solved$ = true then break
15                 $c_{best}.th = \max(n.th, q_2/\delta, \beta)/w_{(n,c_{best})}$
16             else if $n$ is labeled by SUM variable then
17                 $q$ = Sum($n$), $c_{best}$ = UnsolvedChild($n$)
18                 if $q < n.th$ or $n.solved$ = true then break
19                 $c_{best}.th = (n.th - q + q(c_{best}))/w_{(n,c_{best})}$
20             RBFS($c_{best}$)
21     else if $n$ is an AND node then
22         while true do
23             $q$ = Prod($n$)
24             if $q < n.th$ or $n.solved$ = true then break
25             $c_{best}$ = UnsolvedChild($n$)
26             $c_{best}.th = q(c_{best}) \cdot (n.th/q)$
27             RBFS($c_{best}$)
28     SaveCache(Context(n), $q$, $n.solved$)
29     return $q$
```

Lower Bounds

The q-value $q(n)$ which represents an upper bound is typically provided by the heuristic evaluation at node n (i.e., the WMB$_u$ value). In the following, we elaborate on a WMB based approach for computing the initial lower bounding l-values of the nodes. For simplicity, let us assume that node n roots a conditioned summation subproblem and we want to lower bound the corresponding partition function Z below n. Following previous work (Liu and Ihler 2011), a simple way to compute a lower bound on Z is to apply the WMB scheme on the subproblem rooted by n but using *negative* instead of positive weights, which we denote hereafter by WMB$_l$. Specifically, for a given variable X_k, its mini-buckets Q_{kr} are assigned a *negative* weight $w_{kr} < 0$, all except for one which gets a non-negative weight such that $\sum_r w_{kr} = 1$. The intuition behind the use of negative weights is to facilitate the reversed Hölder's inequality which in turn guarantees that eliminating all variables of the subproblem (and corresponding mini-buckets) using the corresponding power sum operators yields a lower bound $\hat{Z}$ on Z (Liu and Ihler 2011).

While the WMB$_l$ scheme is sound for sum-inference, applying it to Eq. 1 does not necessarily yield an overall lower bound on the optimal MMAP value. This could be problematic during search especially if we want to use lower bound values to initialize the l-values of the nodes labeled by MAP variables. Therefore, to ensure correct initial l-values for all nodes in the search space we need to replace $\max$ by $\min$ in Eq. 1 and therefore apply the WMB$_l$ scheme to the expression: $\min_{\mathbf{x}_M} \sum_{\mathbf{x}_S} \prod_{\alpha \in F} \psi_\alpha(\mathbf{x}_\alpha)$. Clearly, each node in the search space can initialize its l-value by applying the WMB$_l$ scheme dynamically on the subproblem below it. However, for efficiency, we can pre-compile the WMB$_l$ scheme along the same ordering used for generating the upper bounding heuristic values (via the WMB$_u$ scheme).

Although this approach typically yields loose initial lower bounds, we will show in our experiments that exploring further the search space associated with the conditioned summation subproblems and revising the l-values of the nodes in a bottom-up manner can potentially generate improved global lower bounds, in some cases much faster than those methods that rely on exact conditioned likelihood evaluations.

The Search Algorithm

Algorithm 4 shows the pseudo-code of RBFAOO+. Besides the q- and l-values, each node n is associated with a *threshold* denoted by $n.th$ which is used to drive the search in a depth-first like manner (Korf 1993). When RBFAOO+ starts solving a problem, the threshold of the root node is set to 0. If RBFAOO+ does not meet this threshold, the problem is proven to have no solution. Otherwise, RBFAOO+ returns the optimal MMAP value of the problem.

Function $\texttt{RBFS}(n)$ traverses the subtree rooted at n in a depth-first like manner. It calculates either an optimal solution value or an upper bound and checks if the termination condition is satisfied (i.e., $q(n) < n.th$). Namely, if $q(n) < n.th$ this indicates that examining n's subtree is no longer the best strategy. If the solution optimality is guaranteed at n, the flag $n.solved$ is set to **true**.

After selecting the current best node n, the algorithm expands n by generating its successors and initializes the corresponding q- and l-values using the $h(n)$ and $lb(n)$ functions, respectively. The latter could be implemented using the WMB$_u$ and WMB$_l$ schemes. Then, a bottom up propagation step updates the values of the nodes along the current path from n to the root, which results in improved global upper and lower bounds (see also lines 1-14 and 15-24 in Algorithm 1 for expansion and propagation, respectively). During propagation, the q-values are updated in the usual way: by maximization or summation for OR nodes (labeled by MAP or sum variables), and by multiplication for AND nodes, respectively. Clearly, the revised q-value $q(s)$ of the root node s is the current best global upper bound. The l-values of the OR nodes labeled by sum variables are updated by summation, while AND nodes update their l-values by multiplication. For OR nodes labeled by MAP variables we only consider the l-value of the AND child on the current path together with the corresponding arc weight. This gives us the lower bound associated with the current path. Then the algorithm reports the best global bounds found so far.

At an OR node labeled by a MAP variable, $\texttt{RBFS}(n)$ may find a suboptimal solution. In this case, $n.solved$ is still set to **false** and $\texttt{RBFS}(n)$ continues examining other children until it finds an optimal solution at n. Because the solution value found so far is still smaller than the optimal one (we solve a maximization problem), $\texttt{RBFS}(n)$ uses that solution value (maintained by β in line 13) to prune away unpromising branches and to adjust the threshold. When $\texttt{RBFS}(n)$ selects the best child c_{best}, it examines c_{best} with a new threshold. Specifically, $c_{best}.th$ is set to dividing the weight between n and c_{best} from the maximum of:

1. The current threshold for n.
2. The second largest upper bound q_2 amongst the unsolved children of n. This indicates when the current second best child becomes the best one. Notice that q_2 includes the corresponding arc weight (line 6 in Algorithm 2). Additionally, q_2 is divided by the *overestimation rate* $\delta \geq 1$ to avoid an excessive number of backtracks to n.
3. The current best solution value β at n.

At AND nodes which are labeled by either MAP or summation variables, $c_{best}.th$ is set to the product of c_{best}'s q-value and the gap between $n.th$ and the total q-value of n's children. If $q(c_{best}) < c_{best}.th$, then $q(n) < n.th$ also holds.

At OR nodes labeled by summation variables, $\texttt{RBFS}$ may in principle have to examine all of n's children. In this case, c_{best} is selected as the one having the largest q-value amongst n's unsolved children. Its threshold is then set to the sum between $q(c_{best})$ and the gap between n's threshold and n's current q-value which is calculated by summation. The intuition behind setting thresholds for OR summation nodes is to diversify the search below these nodes and allow the algorithm to backtrack from the summation subproblems. This in turn could potentially lead to finding tighter bounds.

Finally, based on previous work (Kishimoto and Marinescu 2014), we can show that:

Theorem 1 (correctness and completeness)**.** *Given a graphical model $\mathcal{M}$ algorithm RBFAOO+ is sound and complete.*

We also consider a simple variation of RBFAOO+ which we denote by RBFAOO-. The idea is to forgo setting a threshold for OR nodes labeled by summation variables, thus forcing the algorithm to search completely the conditioned summation subproblems. Specifically, RBFAOO- is obtained by replacing line 19 in Algorithm 4 with $c_{best}.th = 0$.

domain	#	n	k	w_c^* (avg)	w_s^* (avg)
grid	75	144–2500	2	16–376 (96)	4–24 (8)
pedigree	110	334–1289	3–7	18–300 (97)	3–28 (7)
promedas	50	453–1849	2	11–490 (120)	3–18 (6)

Table 1: Benchmark statistics.

Experiments

We evaluate empirically the anytime performance of the proposed algorithms RBFAOO+ and RBFAOO- against recent state-of-the-art anytime bounding schemes on benchmark problems generated from standard probabilistic inference networks (Elidan, Globerson, and Heinemann 2012). The competing algorithms, are the earlier AAOBF, LAOBF and LnDFS, respectively, which are the current best-performing

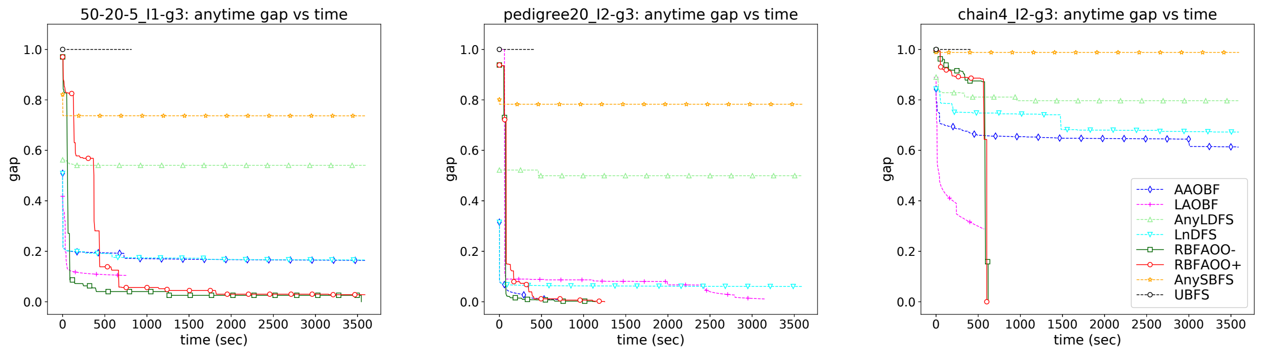


Figure 2: Anytime gap between upper and lower bounds for selected problem instances. Time limit 1 hour.

algorithm	grid	pedigree	promedas
upper bounds (1min/10min/60min)			
AAOBF	57/**75/75**	98/108/108	36/**49/49**
LAOBF	58/**75/75**	99/**110/110**	34/**49/49**
LnDFS	**69/75/75**	104/108/108	**44/49/49**
UBFS	59/**75/75**	103/**110/110**	35/**49/49**
AnyLDFS	7/10/14	0/0/0	20/24/28
AnySBFS	18/19/23	6/6/6	29/32/33
RBFAOO+	53/**75/75**	**105/110/110**	34/**49/49**
RBFAOO-	53/**75/75**	**105/110/110**	34/**49/49**
lower bounds (1min/10min/60min)			
AAOBF	57/**75/75**	98/108/108	36/**50/50**
LAOBF	56/**75/75**	77/92/95	32/49/49
LnDFS	69/**75/75**	104/108/108	44/**50/50**
UBFS	0/0/0	0/0/0	0/0/0
AnyLDFS	**75/75/75**	**110/110/110**	**50/50/50**
AnySBFS	**75/75/75**	**110/110/110**	**50/50/50**
RBFAOO+	53/**75/75**	105/**110/110**	34/**50/50**
RBFAOO-	53/**75/75**	105/**110/110**	34/**50/50**

Table 2: Number of instances with non-trivial upper and lower bounds at 1 min, 10 min and 60 min time intervals.

algorithm	grid	pedigree	promedas
tightest upper bound			
AAOBF	0/25/29	3/16/16	1/13/15
LAOBF	5/34/32	6/26/24	3/19/19
LnDFS	10/6/11	4/3/4	8/8/9
UBFS	11/26/23	25/24/20	**25/35/26**
AnyLDFS	5/0/0	1/0/0	5/2/2
AnySBFS	5/1/0	1/0/0	1/0/1
RBFAOO+	**36**/57/61	31/43/50	4/21/24
RBFAOO-	34/**62/64**	**66/68/78**	6/24/**35**
tightest lower bound			
AAOBF	18/41/38	**47/60/50**	**21/30/33**
LAOBF	11/38/35	38/47/48	10/24/25
LnDFS	**33**/32/33	26/23/23	18/18/20
UBFS	0/0/0	0/0/0	0/0/0
AnyLDFS	5/0/0	5/2/2	8/3/1
AnySBFS	1/0/0	0/0/0	0/0/0
RBFAOO+	0/31/34	0/6/10	0/19/21
RBFAOO-	20/**45/52**	7/17/23	0/21/22

Table 3: Number of instances for which an algorithm found the tightest upper bound and the tightest lower bound, respectively, at 1 min, 10 min and 60 min time intervals.

best+first-search hybrid schemes for providing anytime deterministic lower and upper bounds on the optimal MMAP value (Marinescu et al. 2017). For reference, we also considered UBFS which is a recent bounding scheme for MMAP that is based on best-first search (Lou, Dechter, and Ihler 2018). However, unlike our algorithms, UBFS provides anytime upper bounds only. In addition, we also run two recent stochastic anytime search algorithms for bounding MMAP called AnyLDFS and AnySBFS, respectively (Marinescu, Dechter, and Ihler 2018). In contrast to the previous algorithms, these schemes compute anytime probabilistic upper and lower bounds. For RBFAOO+ and RBFAOO- we set the overestimation parameter δ to 2.0 and their garbage collection scheme replaced $R = 30\%$ of the cache entries. The cutoff parameter θ used by LAOBF to trigger the depth-first lookaheads was set to 1000, while AnySBFS and AnyLDFS were run with the default parameters specified in (Marinescu, Dechter, and Ihler 2018). All competing algorithms used the same heuristic function WMB$_u(i)$ with the i-bound set to 10. Similarly, we used WMB$_l(10)$ to pre-compile the initial lower bounds. The time limit was set to 1 hour and we ran all algorithms with a 20GB of RAM memory limit.

Our benchmark set includes 3 standard problem domains from grid networks (`grid`), genetic linkage analysis (`pedigree`), and medical diagnosis expert systems (`promedas`). Since the original problems are pure MAP tasks, we generated 5 synthetic MMAP instances for each pure MAP instance by randomly selecting 50% of the variables as MAP variables as suggested previously in (Lee et al. 2016; Marinescu et al. 2017). Therefore, we created 75 grid, 110 pedigree, and 50 promedas MMAP instances. Table 1 shows the typical ranges of the problem instance parameters where n is the number of variables, k is the domain size, w_c^* is the constrained induced width and w_s^* is the induced width of the conditioned summation subproblem (we also include in parenthesis the average values of the latter two measures). Note that these instances were generated so that most of them

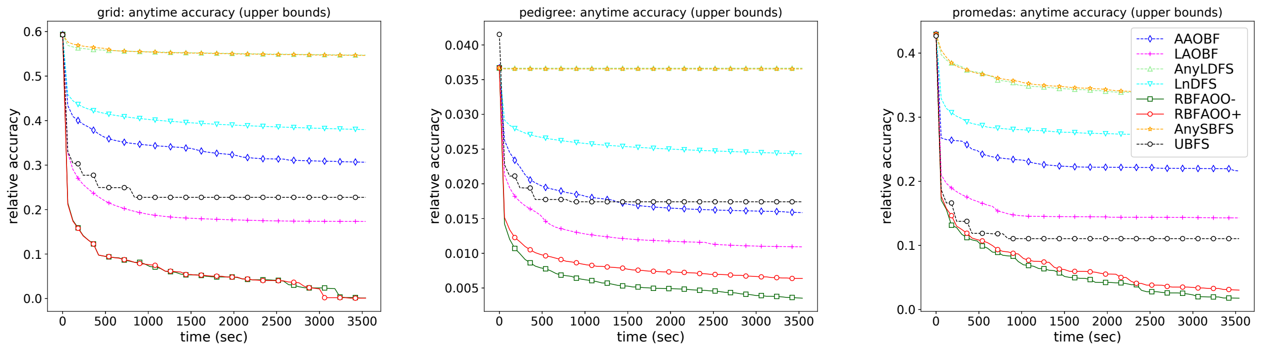


Figure 3: Average relative accuracy for upper bounds. Time limit 1 hour.

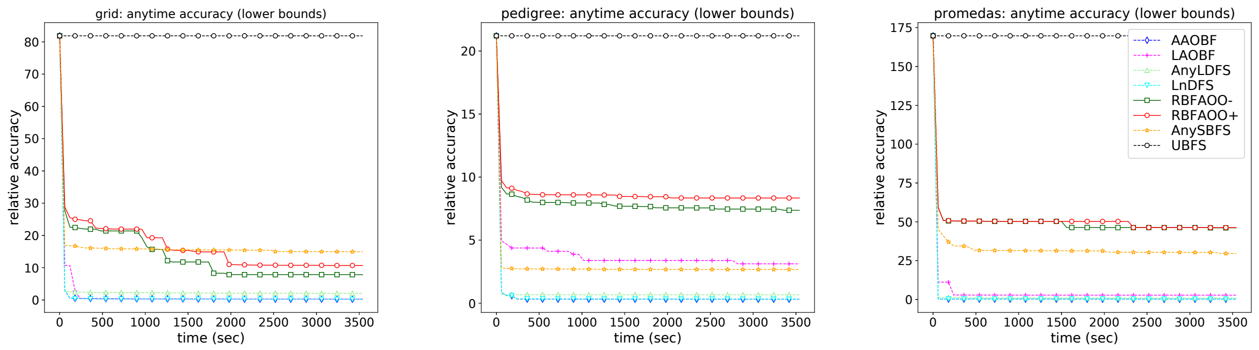


Figure 4: Average relative accuracy for lower bounds. Time limit 1 hour.

could not be solved optimally within 1 hour and therefore be relevant to anytime bounding schemes.

Responsiveness The responsiveness characterizes how fast an algorithm can discover non-trivial bounds. We require all algorithms to produce upper bounds other than the initial bounds from the heuristic. Similarly, we require the lower bounds found to be different from the default one $-\infty$. Table 2 shows the number of instances for which each algorithm found non-trivial upper (top) and lower (bottom) bounds within 1 minute, 10 minutes and 1 hour, respectively. The best performance points are highlighted. We can see that RBFAOO+/RBFAOO- are competitive with the previous deterministic approaches AAOBF, LAOBF and LnDFS as they all discover non-trivial upper and lower bounds on most of the problem instances, across all time bounds.

Results on individual instances Figure 2 plots the gap between the anytime upper and lower bounds obtained on three selected problem instances, one from each benchmark domain. More specifically, we define the gap at time t as $gap_t = \frac{\log(l_t)-\log(u_t)}{\log(l_t)}$, where u_t and l_t are the upper and respectively the lower bounds obtained at time t. Notice that we normalize the gap value in order to be between 0 and 1. Clearly, if $u_t = l_t$ then $gap_t = 0$. Alternatively, if no solution was found, we assume that $gap_t = 1$ (this is typically the case for UBFS because it doesn't compute lower bounds). We can see that both RBFAOO+ and RBFAOO- start off with relatively loose lower bounds (the gap is close to 1). However, as the search progresses, they compute more accurate bounds and therefore are able to tighten the gap considerably compared with the other algorithms. For example, on the grid instance 5-20-5_I1, both RBFAOO+ and RBFAOO- found much tighter bounds with RBFAOO- proving optimality in less than 1 hour, UBFS and LAOBF ran out of memory after nearly 15 minutes, while AAOBF, LnDFS, AnyLDFS and AnySBFS found relatively poor quality bounds thus yielding much larger gaps.

Quality of the bounds Table 3 summarizes the winners in terms of finding the tightest upper (top) and lower (bottom) bounds within a given time bound of 1 minute, 10 minutes and 1 hour, respectively. We can see that RBFAOO+/RBFAOO- typically find the tightest upper bounds, especially for longer time horizons. This is because both algorithms explore effectively the search space associated with the summation subproblems which allows them to propagate the heuristic values associated with the summation nodes as well. This eventually translates into tighter global upper bounds. In contrast, the lower bounds produced by RBFAOO- are slightly tighter than those produced by

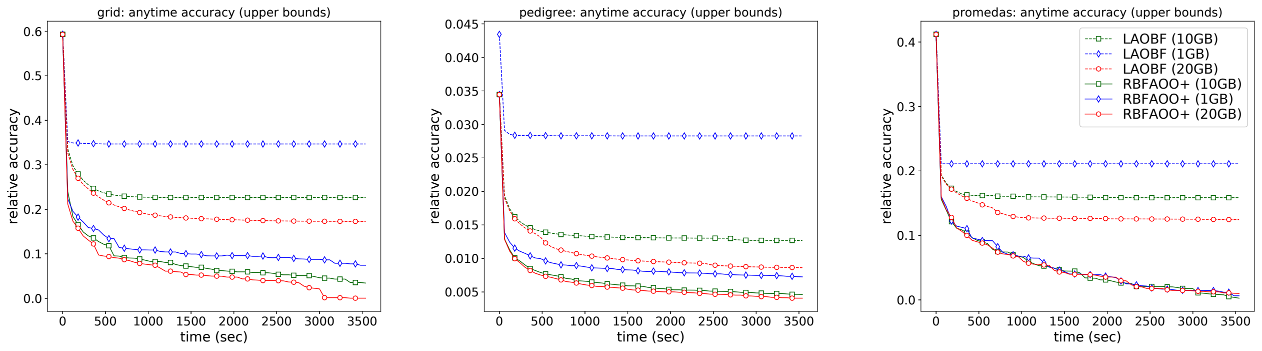


Figure 5: Average relative accuracy for upper bounds. Time limit 1 hour. Memory limits 1GB, 10GB and 20GB.

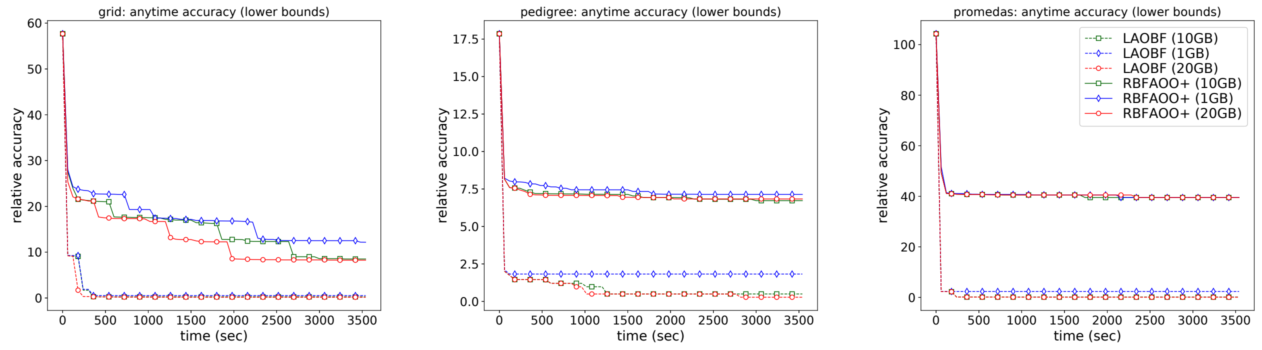


Figure 6: Average relative accuracy for lower bounds. Time limit 1 hour. Memory limits 1GB, 10GB and 20GB.

RBFAOO+, but overall they both are on average inferior to those produced by the deterministic bounding schemes. This is because unlike AAOBF, LAOBF, and LnDFS which compute the lower bounds by solving exactly the summation subproblems, RBFAOO+/RBFAOO- compute the lower bounds by propagating local lower bounds associated with the summation nodes (l-values). The latter are typically quite loose compared with the corresponding exact values.

In Figures 3 and 4 we plot the average relative accuracy with respect to the tightest upper (resp. lower) bounds as a function of time for all three domains and for all competing algorithms. For a given problem instance and algorithm we compute the relative accuracy at time t as $\left|\frac{\log(u_t)-\log(u^*)}{\log(u^*)}\right|$ and $\left|\frac{\log(l_t)-\log(l^*)}{\log(l^*)}\right|$, where u_t (resp. l_t) is the current upper (resp. lower) bounds and u^* (resp. l^*) is the tightest upper (resp. lower) bound found for that instance. If an algorithm did not find a lower bound at time t then we compute the relative accuracy as $\left|\frac{2*\log(l_-)-\log(l^*)}{\log(l^*)}\right|$, where l_- is the worst lower bound for that particular instance. If an algorithm ran out of memory at time t before the time limit then we consider u_t and l_t for all time steps between t and the limit. In summary, we can see that both RBFAOO+ and RBFAOO- dominate clearly all the other competitors in terms of upper bounds. In contrast, the lower bounds produced by RBFAOO- are slightly tighter than those produced by RBFAOO+, but overall both are on average inferior to those produced by the deterministic bounding schemes. The probabilistic schemes AnyLDFS and AnySBFS have a relatively large overhead associated with bounding each conditioned summation subproblem which prevents them to explore a large search space and thus improve the global upper bound.

algorithm	grid	pedigree	promedas
# instances out of memory (1GB/10GB/20GB)			
AAOBF	55/28/24	89/29/22	28/4/1
LAOBF	59/48/44	102/89/71	35/34/30
LnDFS	16/**0**/**0**	17/5/2	1/**0**/**0**
UBFS	61/53/52	103/102/99	35/34/33
AnyLDFS	57/7/**0**	101/11/**0**	50/9/**0**
AnySBFS	19/7/3	60/20/10	13/7/4
RBFAOO+	**0/0/0**	**0/0/0**	**0/0/0**
RBFAOO-	**0/0/0**	**0/0/0**	**0/0/0**

Table 4: Number of instances for which an algorithm ran out of memory at 1GB, 10GB and 20GB memory limits.

Impact of limited memory In Table 4 we summarize the behavior of all competing algorithms under different memory requirements. In particular, we show the number of instances on which the respective algorithm ran out of memory, for 1GB, 10GB and 20GB memory limits. We can see that both RBFAOO+ and RBFAOO- are extremely robust in terms of memory usage as they use the available memory most effectively to improve the bounds. In contrast, the other algorithms ran out of memory on significantly more instances, especially at the smallest memory bound. In Figures 5 and 6 we plot the anytime average gap with respect to the tightest upper and lower bounds, when using 1GB, 10GB and 20GB of memory, respectively. For our purpose, we compare RBFAOO+ against LAOBF which was one of the best performing hybrid schemes (we obtained similar results with RBFAOO- but we omit them for space reasons). When looking at the quality of the upper bounds, we can see that RBFAOO+ hardly suffers from any performance degradation as the memory becomes more restricted. On the other hand, the impact of restricted memory is more prominent for LAOBF especially at the 1GB limit where it computes relatively weak bounds. Finally, in terms of lower bounds, LAOBF produces the tightest bounds regardless of the memory used, whereas RBFAOO+ improves slightly the bounds as more memory is available.

Conclusion

We introduced a recursive best-first AND/OR search method to compute anytime upper and lower bounds on the optimal MMAP value using limited memory. Our approach overcomes a major limitation of existing memory-intensive search-based bounding schemes for MMAP and therefore it can handle successfully difficult MMAP instances with very large search spaces. Our extensive empirical evaluation on various benchmarks demonstrates the effectiveness of the new algorithms compared with existing state-of-the-art deterministic and probabilistic bounding schemes. While we focused here on computing deterministic bounds, we believe that our approach can be extended to compute probabilistic bounds as well, but we leave this as future work. Another important direction of research we plan to pursue in the future is to explore parallel search schemes for MMAP.

Acknowledgements This work was supported in part by NSF grants IIS-1526842 and IIS-1254071, and by contracts FA8750-14-C-0011 and W911NF-18-C-0015 under the DARPA PPAML and World Modelers programs. We are grateful to the reviewers for their helpful comments.

References

Dechter, R., and Mateescu, R. 2007. AND/OR search spaces for graphical models. *Artificial Intelligence* 171(2-3):73–106.

Dechter, R., and Rish, I. 2003. Mini-buckets: A general scheme of approximating inference. *Journal of ACM* 50(2):107–153.

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

Dechter, R. 2013. *Reasoning with Probabilistic and Deterministic Graphical Models: Exact Algorithms*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Elidan, G.; Globerson, A.; and Heinemann, U. 2012. PASCAL 2011 probabilistic inference challenge. http://www.cs.huji.ac.il/project/PASCAL/.

Howard, R., and Matheson, J. 2005. Influence diagrams. *Decision Analysis* 2(3):127–143.

Kask, K., and Dechter, R. 2001. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence* 129(1-2):91–131.

Kishimoto, A., and Marinescu, R. 2014. Recursive best-first AND/OR search for optimization in graphical models. In *Uncertainty in Artificial Intelligence (UAI)*, 400–409.

Korf, R. 1993. Linear-space best-first search. *Artificial Intelligence* 62(1):41–78.

Lee, J.; Marinescu, R.; Dechter, R.; and Ihler, A. 2016. From exact to anytime solutions for marginal MAP. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1749–1755.

Liu, Q., and Ihler, A. 2011. Bounding the partition function using Hölder's inequality. In *International Conference on Machine Learning (ICML)*, 849–856.

Liu, Q., and Ihler, A. 2013. Variational algorithms for marginal MAP. *Journal of Machine Learning Research* 14:3165–3200.

Lou, Q.; Dechter, R.; and Ihler, A. 2018. Anytime anyspace AND/OR best-first search for bounding marginal MAP. In *AAAI Conference on Artificial Intelligence (AAAI)*, 860–867.

Marinescu, R.; Lee, J.; Dechter, R.; and Ihler, A. 2017. Anytime best+depth-first search for bounding marginal MAP. In *AAAI Conference on Artificial Intelligence*, 1749–1755.

Marinescu, R.; Dechter, R.; and Ihler, A. 2014. AND/OR search for marginal MAP. In *Uncertainty in Artificial Intelligence (UAI)*, 563–572.

Marinescu, R.; Dechter, R.; and Ihler, A. 2015. Pushing forward marginal MAP with best-first search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 696–702.

Marinescu, R.; Dechter, R.; and Ihler, A. 2018. Stochastic anytime search for bounding marginal MAP. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 5074–5081.

Maua, D., and Campos, C. D. 2012. Anytime marginal MAP inference. In *International Conference on Machine Learning*, 1471–1478.

Nagai, A. 1999. A new depth-first search algorithm for AND/OR trees. Master's thesis, Department of Information Science, University of Tokyo.

Park, J. 2002. MAP complexity results and approximation methods. In *Uncertainty in Artificial Intelligence (UAI)*, 388–396.

Zobrist, A. L. 1970. A new hashing method with applications for game playing. Technical report, Department of Computer Science, University of Wisconsin, Madison.