Pushing Forward Marginal MAP with Best-First Search

Radu Marinescu
IBM Research – Ireland
radu.marinescu@ie.ibm.com

Rina Dechter and **Alexander Ihler**
University of California, Irvine
Irvine, CA 92697, USA
{dechter,ihler}@ics.uci.edu

Abstract

Marginal MAP is known to be a difficult task for graphical models, particularly because the evaluation of each MAP assignment involves a conditional likelihood computation. In order to minimize the number of likelihood evaluations, we focus in this paper on best-first search strategies for exploring the space of partial MAP assignments. We analyze the potential relative benefits of several best-first search algorithms and demonstrate their effectiveness against recent branch and bound schemes through extensive empirical evaluations. Our results show that best-first search improves significantly over existing depth-first approaches, in many cases by several orders of magnitude, especially when guided by relatively weak heuristics.

1 Introduction

Graphical models provide a powerful framework for reasoning with probabilistic and deterministic information. These models use graphs to capture conditional independencies among variables, allowing a concise representation of knowledge as well as efficient graph-based query processing algorithms. Combinatorial maximization, or maximum *a posteriori* (MAP) tasks arise in many applications and often can be efficiently solved by search schemes.

The marginal MAP problem distinguishes between maximization variables (called MAP variables) and summation variables (the others). Marginal MAP is NP$^{\text{PP}}$-complete [Park, 2002]; it is difficult not only because the search space is exponential in the number of MAP variables, but also because evaluating the probability of any full instantiation of the MAP variables is PP-complete [Roth, 1996]. Algorithmically, its difficulty arises in part from the fact that the variable elimination operations (max and sum) must be applied in a constrained, often far more costly order.

Up to now, search-based marginal MAP solvers have been based on depth-first branch and bound (DFBnB). Searching in a best-first manner is known to be superior, but requires more memory than depth-first search and is therefore less popular for graphical models. Yet, memory effective best-first search algorithms such as *recursive best-first search* [Korf, 1993] have appeared in the context of pure optimization tasks (MAP/MPE) in graphical models [Marinescu and Dechter, 2009b; Kishimoto and Marinescu, 2014]. Their potential for the marginal MAP task is explored here for the first time to the best of our knowledge.

Contributions: Our paper presents and explores the power of several best-first search schemes for solving marginal MAP queries. We discuss the potential benefit of best-first search and demonstrate the effectiveness of several variants against state of the art methods, particularly a recent depth-first branch and bound approach. Through extensive empirical evaluations we show not only orders-of-magnitude improvements over the state of the art, but also the ability to solve problem instances well beyond the reach of previous approaches.

In Section 2 we provide background and review earlier work. Section 3 describes and analyzes the algorithms, Section 4 provides empirical evaluation and Section 5 concludes.

2 Background

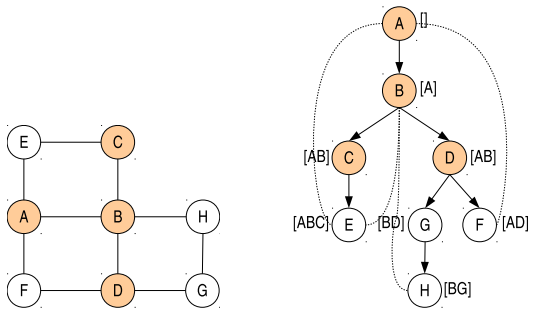
A *graphical model* is a tuple $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, where $\mathbf{X} = \{X_i : i \in V\}$ is a set of variables indexed by set V and $\mathbf{D} = \{D_i : i \in V\}$ is the set of their finite domains of values. $\mathbf{F} = \{\psi_\alpha : \alpha \in F\}$ is a set of discrete positive real-valued local functions defined on subsets of variables, where we use $\alpha \subseteq V$ and $\mathbf{X}_\alpha \subseteq \mathbf{X}$ to indicate the *scope* of function ψ_α, i.e., $\mathbf{X}_\alpha = var(\psi_\alpha) = \{X_i : i \in \alpha\}$. The function scopes yield a *primal graph* whose vertices are the variables and whose edges connect any two variables that appear in the scope of the same function. The graphical model $\mathcal{M}$ defines a factorized probability distribution on $\mathbf{X}$, namely:

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

where 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) problem 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 $\mathbf{X}_S$; this yields the expression:

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



(a) Primal graph (b) Pseudo tree

Figure 1: A simple graphical model.

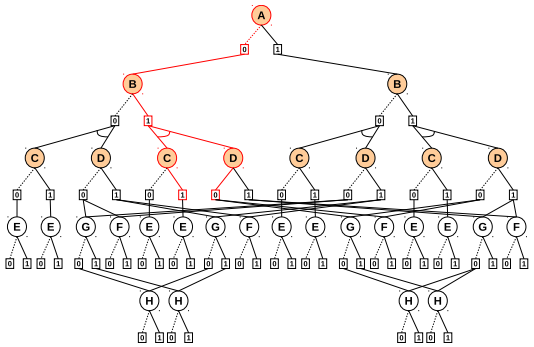


Figure 2: AND/OR search graph.

If $\mathbf{X}_S = \emptyset$ then the problem is the maximum *a posteriori* (MAP) inference (also known as MPE). The marginal MAP problem is significantly more difficult as noted earlier. Practically speaking, the main difficulty arises because the max and sum operators in Eq. (1) do not commute, which restricts the elimination orders to those in which all variables $\mathbf{X}_S$ are eliminated (by summation) before any variable in $\mathbf{X}_M$ can be eliminated (by maximization).

2.1 AND/OR Search Spaces

Significant recent improvements in search for marginal MAP inference have been achieved by using AND/OR search spaces, which often capture problem structure far better than standard OR search methods [Marinescu *et al.*, 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, valid)** *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 included 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. A pseudo tree $\mathcal{T}$ of G is* valid for MAP variables $\mathbf{X}_M$ *if $\mathcal{T}$ restricted to $\mathbf{X}_M$ forms a connected* start pseudo tree *having the same root as $\mathcal{T}$.*

Example 1 *Figure 1(a) shows the primal graph of a simple graphical model with 8 bi-valued variables and 10 binary functions. The MAP and SUM variables are $\mathbf{X}_M = \{A, B, C, D\}$ and $\mathbf{X}_S = \{E, F, G, H\}$, respectively. Figure 1(b) displays a valid pseudo tree whose MAP variables form a start pseudo tree.*

Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ with primal graph G and 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 edges 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 $\mathcal{T}$ (i.e., the constrained induced width).

DEFINITION **2 (MMAP solution tree)** *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 also in $\hat{x}_M$.*

Each node n in $C_{\mathcal{T}}$ can be associated with a *value* $v(n)$; for MAP variables $v(n)$ captures 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, respectively), and AND nodes by multiplication.

Example 2 *Figure 2 displays the context minimal AND/OR search graph based on the valid pseudo tree from Figure 1(b) (the contexts are shown next to the pseudo tree nodes). A MMAP solution subtree corresponding to the MAP assignment $(A = 0, B = 1, C = 1, D = 0)$ is shown in red.*

2.2 Earlier Work: Depth-First AOBB for MMAP

Earlier work on MMAP used a heuristic based on an *exact* solution to an *unconstrained* ordering, introduced by Park and Darwiche [2003] and then refined by Yuan and Hansen [2009]. These techniques appear to work quite well when the unconstrained ordering results in a small *induced width*. However, in many situations this is a serious limitation.

More recently, depth-first AND/OR Branch and Bound (AOBB) search algorithms that explore the context minimal AND/OR search graph were shown to be highly effective for marginal MAP [Marinescu *et al.*, 2014], often far superior to the earlier schemes. AOBB for marginal MAP, (sometimes explicitly denoted as AOBB-MMAP) traverses the context-minimal AND/OR search graph in a depth-first manner guided by an upper-bound heuristic function. It interleaves forward expansion of the current partial solution tree $\bar{x}$ with a backward cost revision step that updates node values, until search terminates. When expanding an OR node labeled by a MAP variable, AOBB-MMAP attempts to prune unpromising domain values as part of a branch and bound

scheme. Specifically, it compares the upper bound heuristic function $f(\bar{x})$ of the current partial solution subtree $\bar{x}$ with the current best lower bound L obtained from the best solution seen so far. It prunes the node associated with $\bar{x}$ if $f(\bar{x}) \leq L$. The optimal marginal MAP value (and the corresponding assignment to the MAP variables) is obtained after the root node is fully evaluated.

The effectiveness of AOBB greatly depends on the quality of the upper bound heuristic function that guides the search. In the case of MMAP, the algorithm uses a recently developed weighted mini-bucket (WMB) based heuristic [Dechter and Rish, 2003; Liu and Ihler, 2011] which can be pre-compiled along the reverse order of the pseudo tree. The heuristic is also enhanced by a cost-shifting scheme that can either be applied in a single pass, or iteratively, by passing messages on the corresponding join-graph [Liu and Ihler, 2011; Marinescu *et al.*, 2014]. This yields two schemes denoted by WMB-MM(i) and WMB-JG(i), where parameter i is called the i-bound and controls the accuracy. The corresponding AOBB variants using these bounds, denoted by AOBB-MM(i) and AOBB-JG(i), were evaluated extensively in Marinescu *et al.* [2014], illustrating superiority against earlier state-of-the-art schemes [Park and Darwiche, 2003; Yuan and Hansen, 2009] on a majority of benchmark instances. However, there are many effective search strategies beyond depth-first methods; here we explore *best-first* methods. For marginal MAP, best-first search has significant potential to improve performance, for reasons we discuss next.

3 Best-First Search for MMAP

In this section we introduce several best-first search algorithms for marginal MAP inference. We start with some analysis focusing on pure A*.

3.1 A* for MMAP

Best-first search schemes are known to be superior to other search schemes that traverse the same search space, assuming all algorithms have access to the same heuristic information. Algorithm A* expands only nodes satisfying $f(n) \geq C^*$ (for maximization) where C^* is the cost of the optimal solution and where $f(n) = g(n) + h(n)$ (for sum cost paths), and $h(n)$ is an upper bound of the best cost extension to a solution (or best cost to go) [Dechter and Pearl, 1985]. Any complete search algorithm, and in particular, DFBnB, must explore all the nodes expanded by A*, assuming the same tie breaking rule, and, in addition it may also explore nodes for which $f(n) < C^*$. Therefore, the expected benefit of A* over any other search algorithm, B, depends on the number of extra nodes explored by B having $f(n) < C^*$.

To understand A*'s performance for MMAP inference, we define an OR search space whose nodes are partial assignments to a subset of the MAP variables along a fixed order $X_1, ..., X_m$. To capture the summation operation applied over $\mathbf{X}_S$, we introduce a final solution/goal node denoted s, which extends any full MAP assignment. A solution path in this search space can be denoted by $(x_1, ..., x_m, s)$. Formally,

DEFINITION **3 (OR MMAP search space)** *Let $\mathbf{X}_M$ be the MAP variables of a graphical models $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ and $\mathbf{X}_S = \mathbf{X} \setminus \mathbf{X}_M$. The search space for MMAP has nodes which are partial assignments over $\mathbf{X}_M$, denoted $\bar{x}_{1..j} = (x_1, \ldots, x_j)$. All full MAP assignments have the same child node, s. All the arc-weights of internal nodes are extracted in the usual manner from the functions $\mathbf{F}$. The arc-weight connecting a full MAP assignment $\mathbf{x}_M$ node to s is defined by (denoting by $(n \rightarrow m)$ the arc from n to its child node m in the directed search graph):*

$$w(\mathbf{x}_M \rightarrow s) = \sum_{\mathbf{X}_S} \prod_{\alpha \in F} \psi_\alpha(\mathbf{x}_\alpha | \mathbf{x}_M).$$

It is easy to show that,

PROPOSITION **1** *Given a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ and MAP variables $\mathbf{X}_M$, any search algorithm finding an optimal solution path over its MMAP search space, finds an optimal solution to its MMAP task.*

The most costly operation when traversing even a single solution path in the MMAP search space is when the algorithm expands a full MAP assignment $\bar{x}_{1..m}$, called a *frontier map node*. This is because it needs to evaluate the last arc-weight, which is the cost of this MAP assignment.

Since any expansion of a full MAP assignment is costly, a search algorithm which minimizes such expansions would be more effective. In fact, we argue that A*'s potential for marginal MAP is likely to be more profound than in regular optimization (e.g., for pure MAP) because it can save not only on the number of regular nodes expanded, but much more significantly on the number of expansions of MAP frontier nodes, and therefore the number of summation sub-tasks. It follows from the behavior of A* that,

THEOREM **1** *Let A* be a search algorithm exploring the MMAP search space and let B be any search algorithm. Assuming all algorithms have access to the same heuristic function and use the same tie breaking rule, a) any node expanded by A* will be expanded by B. b) In particular, any MAP frontier node expanded by A* must be expanded by B.* □

We argue that a DFBnB scheme is likely to expand *extra* MAP frontier nodes beyond best-first because, unlike regular optimization where DFBnB is often supplied with a close to optimal solution (e.g., using local search), finding an initial good solution (and thus a good lower bound) is far harder in the context of marginal MAP.

3.2 Best-First AND/OR Search for MMAP

The best-first search algorithms we propose traverse the context minimal AND/OR search graph over all the variables, and therefore explicitly perform the summation computation as part of the AND/OR search. Doing so implies that we use one of the best algorithms for the summation sub-problem, and also allow sharing the summation sub-task computations via caching. The DFBnB algorithm AOBB-MMAP presented in Marinescu *et al.* [2014] explores this space as well.

Algorithm 1 describes the AOBF-MMAP scheme. The algorithm belongs to the AO* family [Nilsson, 1980] and maintains the partially explored context-minimal AND/OR graph $C_\mathcal{T}$ and the best partial solution tree T' that represents an optimal solution of $C_\mathcal{T}$ under the assumption that the tips n of

Algorithm 1: AOBF-MMAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ such that $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$, pseudo tree $\mathcal{T}$, heuristic $h(\cdot)$
Output: Optimal marginal MAP value (and assignment)

1 Insert root n_0 in $C_\mathcal{T}$, where n_0 is labeled by the root of $\mathcal{T}$
2 Initialize $q(n_0) \leftarrow h(n_0)$ and let $T' = \{n_0\}$
3 **while** *true* **do**
4 Select non-terminal tip node n in the best partial tree T'.
5 **if** $tips(T') = \emptyset$ **then break**
  `// Expand`
6 **foreach** *successor* n' *of* n **do**
7 **if** $n' \notin C_\mathcal{T}$ **then**
8 Add n' as child of n in $C_\mathcal{T}$
9 **if** n' *is OR node labeled by* $X \in \mathbf{X}_S$ **then**
10 $q(n') \leftarrow eval(\mathcal{M}|_{T'})$
11 **else** $q(n') \leftarrow h(n')$
  `// Update`
12 **foreach** *ancestor* m *of* n *in* $C_\mathcal{T}$ **do**
13 **if** m *is OR node* **then**
14 $q(m) \leftarrow \max_{n' \in succ(m)} w(m, n') \cdot q(n')$
15 Mark best successor n' of m as $n' = \text{argmax}_{n' \in succ(m)} w(m, n') \cdot q(n')$, maintaining marked successor if still best
16 **else** $q(m) \leftarrow \prod_{n' \in succ(m)} q(n')$
17 Recompute best partial tree T' by following marked arcs from the root n_0
18 **return** $q(n_0)$

$C_\mathcal{T}$ are the terminal nodes with values given by the heuristic $h(n)$ (if labeled by a MAP variable) or by the corresponding conditioned likelihood (if labeled by a sum variable), respectively (see line 10). Initially, $C_\mathcal{T}$ contains the root node n_0 which is an OR node labeled by the root of the pseudo tree $\mathcal{T}$. Then, at each iteration, the algorithm selects a non-terminal leaf node of the partial solution tree T' and expands it by generating its successors. The best partial tree is then revised by backward node value propagation, which sets the values of the leaves in $C_\mathcal{T}$ to its heuristic or conditioned likelihood values. The latter are calculated using procedure $eval(\mathcal{M}|_{T'})$ which solves exactly the summation sub-problem rooted by the respective sum variable (in our implementation we used depth-first AND/OR search with full/adaptive caching [Marinescu and Dechter, 2009b]. The algorithm finishes when there are no leaf nodes in the best partial tree T'. In this case T' represents the optimal MMAP assignment and its value is given by the revised value of the root node.

The complexity of AOBF-MMAP is time and space $O(n \cdot k^{w_o})$, where w_o is the induced width along an ordering consistent with the pseudo-tree.

3.3 Recursive Best-First AND/OR Search

In practice, however, AOBF-MMAP may require an enormous amount of memory, especially on difficult problem instances. More recently, a limited memory best-first search scheme called Recursive Best-First AND/OR search with Overestimation (RBFAOO) was shown to be extremely powerful for pure MAP [Kishimoto and Marinescu, 2014]. Algorithm 2 presents the extension of RBFAOO to marginal MAP.

Algorithm 2: RBFAOO-MMAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ such that $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$, pseudo tree $\mathcal{T}$, heuristic $h(\cdot)$
Output: Optimal marginal MAP value

1 **Procedure** `RBFAOO()`
2 Insert root n_0 in $C_\mathcal{T}$, where n_0 is labeled by the root of $\mathcal{T}$
3 `OrNode(`n_0, 0`)`
4 **return** $q(n_0)$
5 **Function** `OrNode(`n, θ`)`
6 **while** *true* **do**
7 **foreach** *AND child* c *of* n **do**
8 **if** c *is in cache* **then** $q(c) \leftarrow ReadCache(c)$
9 **else** $q(c) \leftarrow h(c)$
10 $q(c) = w(n, c) \cdot q(c)$
11 Update $q(n) \leftarrow \max_{c \in succ(n)} q(c)$ and mark n as solved if the child with the highest q-value is solved
12 **if** $q(n) < \theta$ *or* n *is solved* **then**
13 **break**
14 Identify two children (n_1, n_2) with the two highest q values s.t. $q(n_1) \geq q(n_2) \geq q(others)$ and update threshold as $\theta' = max(\theta, q(n_2))/w(n, n_1)$
15 `AndNode(`n_1, θ'`)`
16 $WriteCache(n, q(n))$
17 **Function** `AndNode(`n, θ`)`
18 **while** *true* **do**
19 **foreach** *OR child* c *of* n **do**
20 **if** c *is in cache* **then** $q(c) \leftarrow ReadCache(c)$
21 **else if** c *is labeled by* $X \in \mathbf{X}_S$ **then**
22 $q(c) \leftarrow eval(\mathcal{M}|_{\bar{x}})$
23 **else** $q(c) \leftarrow h(c)$
24 Update $q(n) \leftarrow \prod_{c \in succ(n)} q(c)$, and mark n as solved if all its children are solved
25 **if** $q(n) < \theta$ *or* n *is solved* **then**
26 **break**
27 Identify an unsolved OR child n_1 and update threshold $\theta' = \theta \cdot q(n_1)/q(n)$
28 `OrNode(`n_1, θ'`)`
29 $WriteCache(n, q(n))$

RBFAOO-MMAP uses a local threshold control mechanism to explore the context minimal AND/OR search graph in a depth first-like manner [Korf, 1993]. Let $q(n)$, called the q-value, be an upper bound of the solution cost at node n, and $\theta(n)$ be the threshold at n indicating the availability of a second best solution cost besides $q(n)$. RBFAOO-MMAP keeps examining the subtree rooted at n until either $q(n) < \theta(n)$ or the subtree is solved optimally. Therefore, it gradually grows the search space by updating the q-values of the internal nodes and re-expanding them, unlike AOBF-MMAP. The algorithm may operate within linear space (no caching). However, for efficiency purposes, it may use a fixed size cache table to store some of the nodes (based on their contexts). At OR nodes, RBFAOO-MMAP may find a sub-optimal solution, which is a lower bound on the conditioned

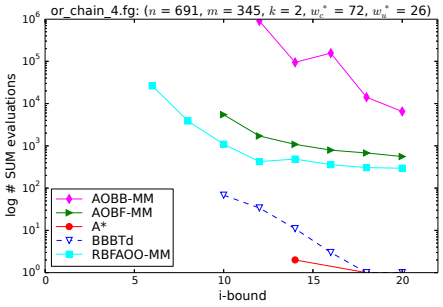


Figure 3: Number of conditional likelihood evaluations for the `or_chain_4.fg` hard instance. Time limit 1 hour.

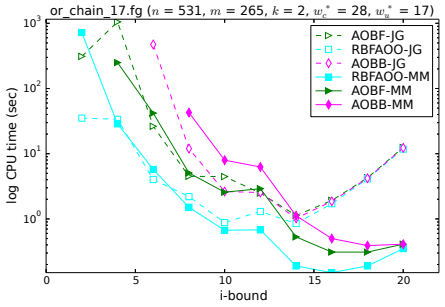


Figure 4: CPU time in seconds for the `or_chain_17.fg` hard instance. Time limit 1 hour.

marginal MAP value. In this case, the algorithm continues to examine other children until it finds an optimal solution at n. Because the solution cost found so far is a lower bound on the optimal one, RBFAOO can use it to prune branches leading to nodes whose q-values are smaller than the solution cost. When RBFAOO-MMAP selects the next best child to expand, it examines it with a new threshold which is updated appropriately for OR and for AND nodes (see lines 14 and 27). Notice also that when expanding an AND node, the algorithm evaluates the conditional likelihood for all OR children that are labeled by sum variables using procedure $eval(\cdot)$ (line 22).

If AOBF-MMAP expands $O(N)$ nodes in the worst case, then RBFAOO-MMAP expands $O(N^2)$ due to node re-expansions. However, in practice, by slightly overestimating the node thresholds (by a small constant δ), RBFAOO-MMAP avoids such a high node re-expansion overhead (see also Kishimoto and Marinescu [2014] for more details).

4 Experiments

We evaluate empirically the proposed best-first search algorithms on problem instances derived from benchmarks used in the PASCAL2 Inference Challenge [Elidan *et al.*, 2012].

We consider two best-first AND/OR search schemes guided by weighted mini-bucket heuristics with moment matching (WMB-MM(i)) and iterative join-graph based cost-shifting (WMB-JG(i)). They are denoted by AOBF-MM and AOBF-JG, respectively. In addition, we consider the recursive best-first search counterparts, denoted by RBFAOO-MM and RBFAOO-JG. These are all MMAP algorithms, but we remove MMAP from the algorithm's name for brevity. The weighted mini-bucket heuristics were generated in a pre-processing phase, prior to search, using uniform weights. We also ran A* search guided by MCTE(i) heuristics which were computed dynamically at each node in the search space. A* explotes the OR search tree defined by the MAP variables.

We compare the best-first search algorithms against each other and against the current state-of-the-art AND/OR branch and bound with weighted mini-bucket heuristics, denoted by AOBB-MM and AOBB-JG, along with the two OR branch and bound schemes guided by MCTE(i) heuristics, denoted by BBBTi and BBBTd [Marinescu *et al.*, 2014]. The pseudo trees guiding the AND/OR algorithms were obtained by a modified min-fill heuristic that constrained the MAP variables to form a start pseudo tree. All algorithms were implemented in C++ (64-bit) and the experiments were run on a 2.6GHz 8-core processor with 80 GB of RAM. The RBFAOO algorithms used a pre-allocated cache of size 10GB. All algorithms were allowed a 1 hour time limit.

Our problem instances were derived from five PASCAL2 benchmarks: `grids` (n-by-n binary grid networks), `segbin` (image segmentation), `protein` (protein side-chain interaction), `promedas` (medical diagnosis expert system), and `pedigree` (genetic linkage analysis). For each network, we generated two marginal MAP problem instances with m MAP variables, as follows: an `easy` instance such that the MAP variables were selected as the first m variables from a breadth-first traversal of a pseudo tree obtained from a hypergraph decomposition of the primal graph (ties were broken randomly) [Marinescu and Dechter, 2009a], and a `hard` instance where the MAP variables were selected uniformly at random. The `easy` instances were designed such that problem decomposition is maximized and the constrained and unconstrained elimination orders are relatively close to each other, thus having comparable induced widths. In contrast, the `hard` instances tend to have very large constrained induced widths. We selected 50% of the variables as MAP variables. In total we evaluated 200 problem instances (20 `easy` and 20 `hard` instances per benchmark).

In all experiments we report the CPU time in seconds and number of nodes visited during search. We also record the problem parameters: number of variables (n), max domain size (k), number of MAP variables (m), and the constrained (w_c^*) and unconstrained (w_u^*) induced widths. We use 'oom' to denote out-of-memory, while '-' means out-of-time.

Conditional likelihood evaluations We compare best-first versus depth-first approaches using the same heuristic, e.g., A* vs. BBBTd (which each use the MCTE heuristic), and AOBF-MM or RBFAOO-MM vs. AOBB-MM (which use the WMB heuristic) on a specific problem instance in Figure 3. For both heuristics, the best-first search schemes evaluate far fewer conditional sum sub-problems, especially at relatively small i-bounds which correspond to weaker heuristics. For example, RBFAOO-MM(12) and AOBF-MM(12) solve over two orders of magnitude fewer summation sub-problems than AOBB-MM(12). (Note that on this problem instance, the MCTE heuristic works very well, but gave poor performance and was unable to solve many of the other instances.)

instance	algorithm	$i=4$		$i=6$		$i=10$		$i=14$		$i=18$		$i=20$	
(n, m, k, w_c^*, w_u^*)		time	nodes	time	nodes	time	nodes	time	nodes	time	nodes	time	nodes
easy grid instances													
50-20-5	AOBB-MM		-		-		-		-	18	576852	4	236597
(400,200,2,30,23)	AOBF-MM		oom		oom	2785	49776304	111	2880234	4	130416	3	84072
	RBFAOO-MM		-		-	821	78576891	60	2631348	6	148816	4	109990
90-20-5	AOBB-MM		-		-	2510	25798937	10	154907	1	18277	1	6745
(400,200,2,30,23)	AOBF-MM		oom		oom	17	259906	1	28744	<1	3231	1	1033
	RBFAOO-MM	1330	172688969	258	37715871	5	639538	1	41912	<1	5253	1	1586
hard grid instances													
50-14-5	AOBB-MM		-		-	111	15689122	1	184242	<1	16126	<1	4248
(196,98,2,37,16)	AOBF-MM		oom	2398	54812995	15	556646	<1	39841	<1	6461	<1	1697
	RBFAOO-MM		-	309	28456443	2	325141	<1	41500	<1	13479	<1	3464
75-16-5	AOBB-MM		-		-		-	608	22468245	9	623549	7	528557
(256,128,2,84,21)	AOBF-MM		oom		oom		oom	7	210304	4	105811	6	148785
	RBFAOO-MM		-		-		-	4	459712	3	374779	7	735433
easy promedas instances													
or_chain_17.fg	AOBB-MM	1345	117992988	42	4185034	3	341352	<1	26663	<1	2550	<1	1140
(531,265,2,20,18)	AOBF-MM	18	719105	6	306513	1	61029	<1	6880	<1	733	<1	341
	RBFAOO-MM	6	1271011	1	292536	<1	72401	<1	8210	<1	970	<1	360
or_chain_22.fg	AOBB-MM		-		-		-		-		-		-
(1044,522,2,69,59)	AOBF-MM		oom		oom		oom		oom		oom		oom
	RBFAOO-MM		-		-		-		-	570	69108302	913	110200946
hard promedas instances													
or_chain_4.fg	AOBB-MM		-		-		-	394	31923310	54	4768363	26	2343346
(691,345,2,72,26)	AOBF-MM		oom		oom	488	15245069	74	2565459	15	576233	11	420072
	RBFAOO-MM		-	3181	328169832	88	8972169	11	1728788	4	677061	4	652188
or_chain_17.fg	AOBB-MM		-		-	22	2196873	5	567511	<1	92971	<1	31770
(531,265,2,28,17)	AOBF-MM		-	143	3736972	7	331471	3	137148	<1	30953	<1	11623
	RBFAOO-MM	318	42057272	24	4084274	1	272656	<1	129827	<1	36854	<1	13258
hard pedigree instances													
pedigree20	AOBB-MM		-		-		-		-		-	15	89147
(437,195,5,58,22)	AOBF-MM	2714	41959199	2940	42216253		oom	41	1096389	13	314660	6	23667
	RBFAOO-MM		-		-	2880	131642655	25	1811067	9	567588	7	73182
pedigree39	AOBB-MM		-		-		-	2632	239991245	1254	101313449	550	48205633
(1272,481,5,44,18)	AOBF-MM		oom		oom		oom	302	5564185	45	1229943	19	446096
	RBFAOO-MM		-		-	2049	292224766	29	5775997	4	476697	6	442276

Table 1: CPU time (sec) and nodes for grid, promedas and pedigree instances. Time limit 1 hour.

Impact of the heuristic quality In Figure 4 we plot results for solving the promedas or_chain_17.fg-hard instance which is quite representative. When looking at the iterative scheme WMB-JG(i), run for 10 iterations, we notice that the corresponding heuristics are most powerful at lower i-bounds because of reduced overhead. As the i-bound increases, its accuracy ceases to offset the computational overhead as the run time of the iterations increase. In this case, WMB-MM(i) is a cost-effective alternative, with reduced overhead for compiling the heuristic.

Comparison with state-of-the-art Table 1 reports the CPU time in seconds and number of nodes expanded by each search algorithm, either depth-first or best-first, on a selection of easy and hard instances from the grid, promedas and pedigree benchmarks. The columns are indexed by the i-bound and the time limit was set to 1 hour. For space reasons, we report results when using WMB-MM(i) heuristics only. The OR search algorithms A*, BBBTi and BBBTd performed poorly and thus are not included. We can see clearly that the recursive best-first AND/OR search scheme is the overall best performing algorithm, especially for relatively small i-bounds which yield relatively inaccurate heuristic estimates. This is important because on extremely difficult problem instances it is likely that we only have access to relatively weak heuristics. For example, on the or_chain_17.fg easy instance, RBFAOO-MM(4) proves optimality in 6 seconds, whereas AOBB-MM(4) finishes in 1345 seconds, respectively. Indeed, the search space explored by RBFAOO-MM(4) is also significantly smaller than that traversed by AOBB-MM(4). Algorithm AOBF-MM(i) is also competitive on most of the instances, however it sometimes suffers from increased computational overhead caused by maintaining its larger search space in memory.

Summary Figure 5 plots the number of problem instances solved from each benchmark (top) and the median CPU time (bottom) as a function of the i-bound. For space reasons, we only show the grid-easy, promedas-hard and pedigree-hard benchmarks. For clarity we report only on the AND/OR search algorithms guided by WMB-MM(i) heuristics. Clearly, the best-first AND/OR search approaches, in particular RBFAOO-MM(i), solve the largest number of instances across i-bounds. On the pedigree instances, AOBF-MM(i) is superior compared with the other schemes at smaller i-bounds which demonstrates the benefit of best-first over depth-first search, when the search space explored actually fits in memory. When the i-bound increases, the heuristics get stronger, reducing the difference between depth-first and best-first search; in this setting depth-first search may find near-optimal solutions quickly and thus, like best-first, will not explore solutions with cost smaller than the optimum. Yet we still see superiority for high i-bound on hard pedigrees and on the selected instances in Table 1.

5 Conclusion

We introduced best-first search schemes for marginal MAP that explore the compact AND/OR context minimal search space for graphical models, guided by pre-compiled weighted mini-bucket with cost-shifting schemes. We explained why best-first search strategies for MMAP can be particularly effective and backed this up by empirical evidence. Specif-

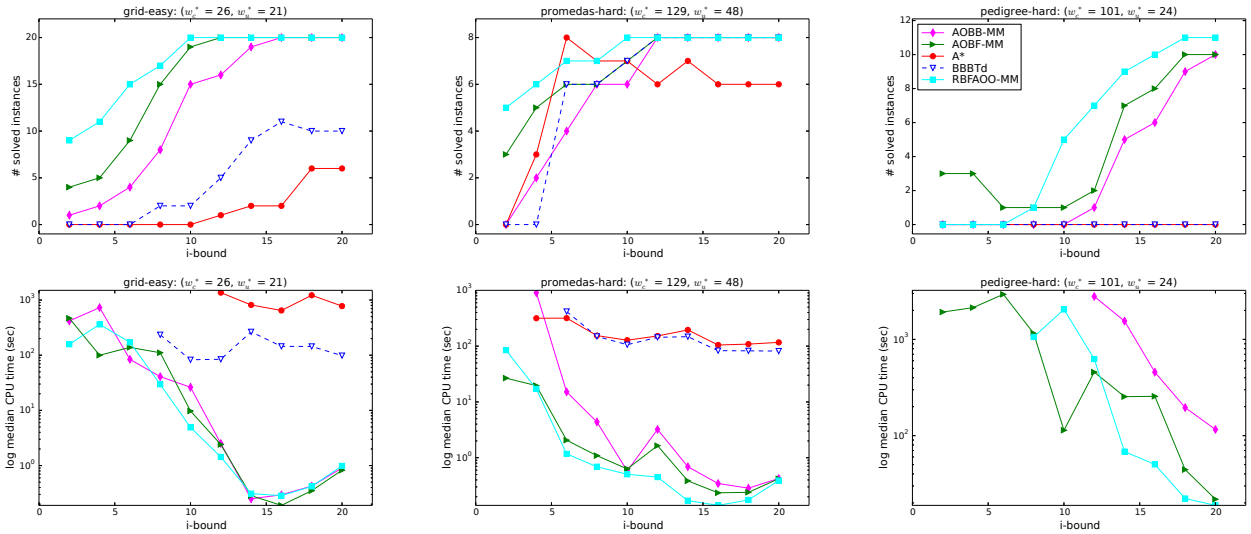


Figure 5: Number of instances solved (top) and median CPU time (bottom) as a function of i-bound for `grid`, `promedas` and `pedigree` instances. The median CPU time is taken over 20 instances in each problem class. Time limit 1 hour.

ically, in an extensive empirical evaluation (some of which was not included for lack of space) we illustrated the superiority of best-first strategies over depth-first search counterparts (e.g., against AOBB-MMAP). In particular, the limited memory recursive best-first AND/OR search scheme, consistently solved more problems given a time-bound (Figure 5) and in many cases was significantly faster, by orders of magnitude, than all depth-first counterparts. It emerges as the superior scheme overall. Our next step would be to extend all these search schemes into anytime algorithms and to explore bounded approximations for the summation sub-tasks.

Acknowledgments

This work was supported in part by NSF grants IIS-1065618 and IIS-1254071, and by the US Air Force under Contract No. FA8750-14-C-0011 under the DARPA PPAML program.

References

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

[Dechter and Pearl, 1985] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *In Journal of ACM*, 32(3):505–536, 1985.

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

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

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

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

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

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

[Marinescu and Dechter, 2009b] R. Marinescu and R. Dechter. Memory intensive AND/OR search for combinatorial optimization in graphical models. *Artificial Intelligence*, 173(16-17):1492–1524, 2009.

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

[Nilsson, 1980] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980.

[Park and Darwiche, 2003] J. Park and A. Darwiche. Solving MAP exactly using systematic search. In *Uncertainty in Artificial Intelligence (UAI)*, pages 459–468, 2003.

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

[Roth, 1996] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[Yuan and Hansen, 2009] C. Yuan and E. Hansen. Efficient computation of jointree bounds for systematic MAP search. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1982–1989, 2009.