Anytime Anyspace AND/OR Best-first Search for Bounding Marginal MAP

Qi Lou
University of California, Irvine
Irvine, CA 92697, USA
qlou@ics.uci.edu

Rina Dechter
University of California, Irvine
Irvine, CA 92697, USA
dechter@ics.uci.edu

Alexander Ihler
University of California, Irvine
Irvine, CA 92697, USA
ihler@ics.uci.edu

Abstract

Marginal MAP is a key task in Bayesian inference and decision-making. It is known to be very difficult in general, particularly because the evaluation of each MAP assignment requires solving an internal summation problem. In this paper, we propose a best-first search algorithm that provides anytime upper bounds for marginal MAP in graphical models. It folds the computation of external maximization and internal summation into an AND/OR tree search framework, and solves them simultaneously using a unified best-first search algorithm. The algorithm avoids some unnecessary computation of summation sub-problems associated with MAP assignments, and thus yields significant time savings. Furthermore, our algorithm is able to operate within limited memory. Empirical evaluation on three challenging benchmarks demonstrates that our unified best-first search algorithm using pre-compiled variational heuristics often provides tighter anytime upper bounds compared to those state-of-the-art baselines.

Introduction

Probabilistic graphical models, including Bayesian networks and Markov random fields, provide a framework for representing and reasoning with probabilistic and deterministic information (Dechter 2013; Dechter, Geffner, and Halpern 2010; Darwiche 2009). Typical inference queries in graphical models include *maximum a posteriori* (MAP) that aims to find an assignment of MAP variables (a.k.a., MAX variables) with the highest value, the *partition function* that is the normalizing constant ensuring a proper probability measure over all variables, and marginal MAP (MMAP) that generalizes the aforementioned two tasks by maximizing over a subset of variables with the remaining variables marginalized.

MMAP arises in many scenarios such as latent variable models (Ping, Liu, and Ihler 2014) and decision-making tasks (Kiselev and Poupart 2014). MMAP has complexity NP^{PP} (Park 2002), commonly believed to be harder than either max inference (NP-complete (Darwiche 2009)) or sum inference (#P-hard (Valiant 1979)), and is even intractable for tree-structured models (Park 2002). The hardness of MMAP is partly due to the fact that max and sum operators do not commute.

Some early works (Park and Darwiche 2003; Yuan and Hansen 2009) solve MMAP exactly based on depth-first branch and bound. Marinescu, Dechter, and Ihler (2014) outperforms the predecessors by using AND/OR search spaces and high-quality variational heuristics. Later, a best-first search variant of Marinescu, Dechter, and Ihler (2014) was proposed and shown empirically superior to the depth-first one given enough memory (Marinescu, Dechter, and Ihler 2015). Because best-first search is memory-intensive, a recursive best-first search algorithm (Marinescu, Dechter, and Ihler 2015) that operates within limited memory was proposed.

However, because of the inherent difficulty of MMAP, searching for exact solutions is generally unpromising. Recent works on MMAP often focus on approximate schemes. Among the approximate schemes, those with deterministic or probabilistic guarantees are of particular interest because they quantify the bounds on the approximation errors. Also, we often favor those approaches with anytime behavior because they allow users to trade computational resources with solution quality. Approximate elimination methods (Liu and Ihler 2011; Dechter and Rish 2003) and closely related variational bounds (Liu and Ihler 2013; Wainwright and Jordan 2008) provide deterministic guarantees. However, these bounds are not anytime; their quality often depends on the memory available, and does not improve without additional amount of memory. Although recent work (Ping, Liu, and Ihler 2015) in the same class has anytime behavior, it is not guaranteed to find optimal solutions unless given enough memory. Some Monte Carlo methods such as one based on random hashing (Xue et al. 2016) provide probabilistic bounds only, while some may not even have this property, e.g., those based on Markov chain Monte Carlo (Yuan, Lu, and Druzdzel 2004; Doucet, Godsill, and Robert 2002). Another algorithm that provides anytime deterministic bounds for MMAP is based on factor set elimination (Mauá and de Campos 2012), but the factor sets it maintains tend to grow very large and thus limit its practical use to problems with relatively small induced widths (Marinescu et al. 2017).

Recent search-based algorithms improve upon their exact predecessors (Marinescu, Dechter, and Ihler 2015; 2014; Otten and Dechter 2012) by introducing weighted heuristics in best-first search (Lee et al. 2016) or by combining best-first search with depth-first search (Marinescu et al. 2017)

to gain the anytime property, and are the state-of-the-art currently. However, those algorithms (Marinescu et al. 2017; Marinescu, Dechter, and Ihler 2015; 2014; Lee et al. 2016) treat the max- and sum- inference separately, and require solving a number of conditional summation problems exactly which is generally intractable.

Although this strategy works reasonably at anytime improvement of lower bounds, it is often slow to improve the associated upper bound. Moreover, it is not always necessary to fully solve an internal summation problem to prove a configuration's sub-optimality. In this paper, we focus our solver on quickly reducing its MMAP upper bound via best-first search, with computation of the internal summation problems integrated in a best-first way as well. When the search encounters an internal summation problem, instead of fully solving it immediately we gradually instantiate its summation variables, which acts to reduce the upper bound of the current candidate MAP (sub-) configuration. This "pure" best-first behavior can potentially solve fewer summation problems to certify the optimal MAP configuration.

Our contributions. This paper presents an anytime anyspace AND/OR best-first search algorithm to improve deterministic upper bounds for MMAP. Our algorithm unifies max- and sum- inference within a specifically designed priority system that aims to reduce upper bound of the optimal solution(s) as quickly as possible, which generalizes the recent best-first search algorithm of Lou, Dechter, and Ihler (2017) for the pure summation task. Our approach avoids some unnecessary exact evaluation of conditional summation problems, yielding significant computational benefits in many cases, as demonstrated by empirical results on three challenging benchmarks compared to existing state-of-the-art baselines on anytime upper bounds.

Background

Let $X = (X_1, \ldots, X_N)$ be a vector of random variables, where each X_i takes values in a discrete domain $\mathcal{X}_i$; we use lower case letters, e.g. $x_i \in \mathcal{X}_i$, to indicate a value of X_i. A graphical model over X consists of a set of factors $\mathcal{F} = \{f_\alpha(X_\alpha) \mid \alpha \in \mathcal{I}\}$, where each factor f_α is defined on a subset $X_\alpha = \{X_i \mid i \in \alpha\}$ of X, called its scope.

We associate an undirected graph $\mathcal{G} = (V, E)$, or *primal graph*, with $\mathcal{F}$, where each node $i \in V$ corresponds to a variable X_i and we connect two nodes, $(i, j) \in E$, iff $\{i, j\} \subseteq \alpha$ for some α. Then,

$$f(x) = \prod_{\alpha \in \mathcal{I}} f_\alpha(x_\alpha)$$

defines an unnormalized probability measure over X.

Let X_{M} be a subset of X called MAX variables, and $X_{\mathrm{S}} = X \setminus X_{\mathrm{M}}$ SUM variables. The marginal MAP task seeks an assignment $x^\star_{\mathrm{M}}$ of X_{M} with the largest marginal probability:

$$x^\star_{\mathrm{M}} = \operatorname*{argmax}_{x_{\mathrm{M}}} \sum_{x_{\mathrm{S}}} f(x)$$

If X_{M} is an empty set, the marginal MAP task reduces to computing the partition function; if X_{S} is empty, it becomes the standard MAP inference task.

AND/OR Search Spaces

An AND/OR search space is a generalization of the standard ("OR") search space, that enables us to exploit conditional independence structure during search (Dechter and Mateescu 2007). The AND/OR search space for a graphical model is defined relative to a *pseudo tree* that captures problem decomposition along a fixed search order.

Definition 1 (**pseudo tree**). *A pseudo tree of a primal graph $\mathcal{G} = (V, E)$ is a directed tree $\mathcal{T} = (V, E')$ sharing the same set of nodes as $\mathcal{G}$. The tree edges E' form a subset of E, and each edge $(i, j) \in E \setminus E'$ are required to be a "back edge", i.e., the path from the root of $\mathcal{T}$ to j passes through i (denoted $i \leq j$).*

If a tree node of a pseudo tree corresponds to a MAX variable in the associated graphical model of the pseudo tree, we call it MAX node, otherwise we call it SUM node. A pseudo tree is called *valid* for an MMAP task if there is *no* MAX variable that is descendant of some SUM variable. Thus, all MAX variables of a valid pseudo tree form a subtree (assuming a dummy MAX root) that contains the root. We only consider valid pseudo trees throughout this paper.

Example. Fig. 1(a) shows the primal graph of a pairwise model. Variables A, B, C are MAX variables, and the rest SUM. Fig. 1(b) shows one valid pseudo tree of the model.

Guided by a pseudo tree, we can construct an AND/OR search tree consisting of alternating levels of OR and AND nodes for a graphical model. Each OR node s is associated with a variable, which we lightly abuse notation to denote X_s; the children of s, $ch(s)$, are AND nodes corresponding to the possible values of X_s. If an OR node is associated with some MAX variable, it is called OR-MAX node. Notions of OR-SUM, AND-MAX, AND-SUM nodes are defined analogously. The root $\emptyset$ of the AND/OR search tree corresponds to the root of the pseudo tree. Let $pa(c) = s$ indicate the parent of c in the AND/OR tree, and $an(c) = \{n \mid n \leq c\}$ indicate the ancestors of c (including itself) in the tree.

In an AND/OR tree, any AND node c corresponds to a partial configuration $x_{\leq c}$ of X, defined by its assignment and that of its ancestors: $x_{\leq c} = x_{\leq p} \cup \{X_s = x_c\}$, where $s = pa(c), p = pa(s)$. For completeness, we also define $x_{\leq s}$ for any OR node s, which is the same as that of its AND parent, i.e., $x_{\leq s} = x_{\leq pa(s)}$.

Definition 2 (**partial solution tree & partial MAP solution tree**). *A partial solution tree T of an AND/OR search tree $\mathcal{T}$ is a subtree satisfying three conditions: (1) T contains the root of $\mathcal{T}$; (2) if an OR node is in T, **at most** one of its children is in T; (3) if an AND node is in T, all of its children **or** none of its children are in T. Moreover, T is called a partial MAP solution tree if T satisfies one additional condition: (4) if an OR-SUM node is in T, this node must have OR-MAX sibling(s). T contains all its siblings and meanwhile T does not contain any of its children.*

The notion of a "partial solution tree" generalizes the notion of a "solution tree" (e.g., Dechter and Mateescu (2007)) which we call a "full solution tree" later on to avoid ambiguity. Any partial solution tree T defines a partial configuration

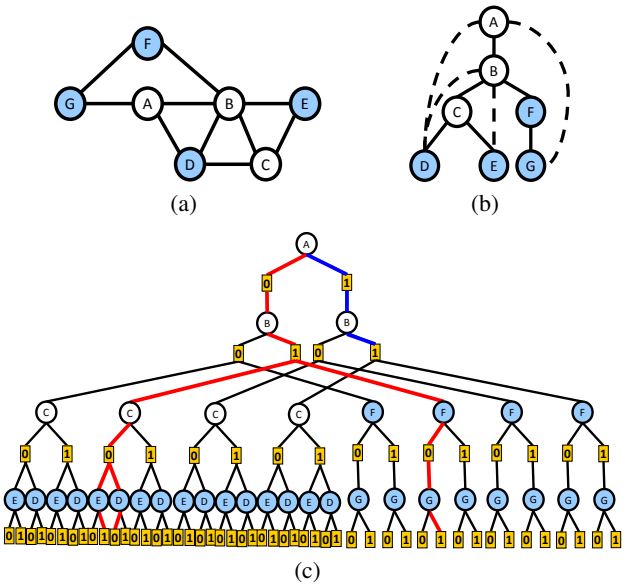


Figure 1: (a) A primal graph of a graphical model over 7 variables (A, B, C are MAX variables and D, E, F, G are SUM variables) with unary and pairwise potential functions. (b) A valid pseudo tree for the primal graph. (c) AND/OR search tree guided by the pseudo tree. One full solution tree is marked red. One partial solution tree that is also a partial MAP solution tree is marked blue.

x_T of X, where $x_T = \cup_{n\in T} x_{\le n}$. If x_T is a complete assignment of X, we call T a full solution tree. A full solution tree corresponds to a complete configuration of all the variables and vice versa. We let $\mathbb{T}$ denote the set of all full solution trees. The associated configuration $x_{T^{\text{M}}}$ of a partial MAP solution tree T^{M} is a partial assignment of X_{M}. If $x_{T^{\text{M}}}$ is a complete assignment X_{M}, we call it a full MAP solution tree. We let $\mathbb{T}^{\text{M}}$ denote the set of all full MAP solution trees.

We also associate a weight w_c with each AND node, defined to be the product of all factors f_α that are instantiated at c but not before:

$$w_c = \prod_{\alpha\in\mathcal{I}_c} f_\alpha(x_\alpha),\ \ \mathcal{I}_c = \{\alpha \mid X_{pa(c)} \in X_\alpha \subseteq X_{an(c)}\}$$

For completeness, we define $w_s = 1$ for any OR node s. It is then easy to see that the product of weights on a path to the root, $g_c = \prod_{a\le c} w_a$ (termed the *cost* of the path), equals the value of the factors whose scope is fully instantiated by x_c. We extend this cost notion to any partial solution tree T by defining g_T as the product of all factors being fully instantiated by x_T. Particularly, the cost of any full solution tree equals the value of its complete configuration.

Example. A full solution tree is marked red in the AND/OR search tree shown in Fig. 1(c). According to the definition, a solution tree corresponds to one full instantiation of all the variables. Its cost is defined as product of weights of all its AND nodes, and equals the value of this configuration in the graphical model. A partial solution tree that happens to be a partial MAP solution tree is marked blue in Fig. 1(c).

We associate a value v_n to any node n, which represents the inference task's value of the search space below node n, and can be defined in a bottom-up recursion. First, if n is an AND node that corresponds to a leaf node of the pseudo tree, we define $v_n = 1$; and for the rest, we have

$$\text{AND node } n\text{: } v_n = \prod_{s\in ch(n)} v_s$$

$$\text{OR node } n\text{: } \quad v_n = \begin{cases} \max\limits_{s\in ch(n)} w_s v_s, & \text{if MAX node } n \\ \sum\limits_{s\in ch(n)} w_s v_s, & \text{if SUM node } n \end{cases} \quad (1)$$

These definitions extend those of Eq. (1)-(2) in Lou, Dechter, and Ihler (2017), to the mixed MAX/SUM elimination of the MMAP task. The value $v_\emptyset$ at the root is the optimum of the MMAP task, i.e., $v_\emptyset = \max_{x_{\text{M}}} \sum_{x_{\text{S}}} f(x)$.

For each node n, we denote V_n as the task value of the model conditioned on $x_{\le n}$, expressed explicitly as:

$$V_n = g_n\, v_n \prod_{s\in branch(n)} v_s \quad (2)$$

where $branch(n)$ is defined as the set of all OR nodes that are siblings of some node $\le n$. In general, for any partial solution tree T, let V_T denote the optimum of the MMAP task for the model conditioned on x_T, we have

$$V_T = g_T \prod_{n\in leaf(T)} v_n \quad (3)$$

where $leaf(T)$ is the set of all leaf nodes of T. Let T_n^{M} be the optimal full MAP solution that contains or extends to n, i.e., $T_n^{\text{M}} = \text{argmax}_{\{T^{\text{M}}\in\mathbb{T}^{\text{M}} | T^{\text{M}}\cup\{n\}\subset T, T\in\mathbb{T}\}} V_{T^{\text{M}}}$. The following proposition reveals the fact that V_n is the total sum of conditioned task values over those full solution trees that contain both T_n^{M} and n:

Proposition 1.

$$V_n \quad = \sum_{\{T\in\mathbb{T} | T_n^{\text{M}}\cup\{n\}\subset T\}} V_T \quad (4)$$

Those quantities and their relation will help us approach the priority defined in our algorithm in the sequel.

Unified Best-first Search

In this section, we will present our main algorithm by first introducing the double-priority system which leads to a simplified version that is A*-like, and then generalizing it to an SMA*-like version (Russell 1992) to operate with limited memory. If the MMAP task corresponds to a pure summation, our algorithm will reduce to that of (Lou, Dechter, and Ihler 2017) using "upper" priority.

Beginning with only the root $\emptyset$, we expand the search tree respecting the pseudo tree structure. As a best-first scheme, we assign a priority to each frontier node on the search tree, then expand the top priority frontier node in each iteration. More precisely, we maintain an explicit AND/OR search tree of visited nodes, denoted $\mathcal{Q}$, whose frontier nodes are denoted *OPEN*. Without loss of generality, we assume *OPEN* only

contains AND nodes. Other nodes of $\mathcal{Q}$ are called *internal* nodes. For an internal node n, *OPEN*(n) denotes the set of descendants of n that are in *OPEN*. A frontier AND node of $\mathcal{Q}$ is *solved* if it corresponds to a leaf node in the pseudo tree. An internal node of $\mathcal{Q}$ is solved if all its children are solved. For an internal OR-MAX node, it suffices to conclude that it is solved if its "best" child $\operatorname{argmax}_{s\in ch(n)} w_s v_s$ is solved.

For each node n in the AND/OR search tree, we make u_n an upper bound of v_n, initialized via pre-compiled heuristic h_n^+ s.t. $v_n \le h_n^+$, and subsequently tightened during search. Given the currently expanded tree $\mathcal{Q}$, we update u_n using information propagated from the frontier, analogous to (1):

$$\text{AND node } n\text{: } u_n = \prod_{s\in ch(n)} u_s$$

$$\text{OR node } n\text{: } \quad u_n = \begin{cases} \max\limits_{s\in ch(n)} w_s u_s, & \text{if MAX node } n \\ \sum\limits_{s\in ch(n)} w_s u_s, & \text{if SUM node } n \end{cases} \tag{5}$$

These values depend implicitly on the search tree $\mathcal{Q}$. Thus, the dynamically updated bounds $U = u_\emptyset$ at the root serve as an anytime bound on the optimum of the MMAP task.

Priority

Our goal is to drive down the global upper bound U as quickly as possible. Intuitively, this can be achieved by expanding the frontier node that affects U most at each iteration. Following this intuition, we will first show how to make connection between a frontier node n and the global upper bound U, which is bridged by the current "best" partial MAP solution tree in $\mathcal{Q}$ that contains or extends to n; we will then establish a double-priority system that marks the most "influential" frontier node as the top priority one.

Analogously to V_n and V_T (see (2), (3)), we define U_n for any node $n \in \mathcal{Q}$ and U_T for any partial solution tree $T \subset \mathcal{Q}$:

$$U_n = g_n\, u_n \prod_{s\in branch(n)} u_s, \qquad U_T = g_T \prod_{n\in leaf(T)} u_n \tag{6}$$

These two quantities are upper bounds of V_n and V_T respectively. Note that U_n and U_T depend on the search tree $\mathcal{Q}$ while V_n and V_T do not. The relation between U_n and U_T is also analogous to that of V_n and V_T stated in (4):

$$U_n = \sum_{\{T\in\mathbb{T}_\mathcal{Q} | T^\mathbb{M}_\mathcal{Q}(n)\cup\{n\}\subset T\}} U_T \tag{7}$$

where $\mathbb{T}_\mathcal{Q} = \{T \cap \mathcal{Q} \mid T \in \mathbb{T}\}$ is the set of the partial solution trees formed by projections of all full solution trees on $\mathcal{Q}$, $\mathbb{T}^\mathbb{M}_\mathcal{Q} = \{T^\mathbb{M} \cap \mathcal{Q} \mid T^\mathbb{M} \in \mathbb{T}^\mathbb{M}\}$ is the set of partial MAP solution trees formed by projections of all full MAP solution trees on $\mathcal{Q}$, and $T^\mathbb{M}_\mathcal{Q}(n) = \operatorname{argmax}_{\{T^\mathbb{M}\in\mathbb{T}^\mathbb{M}_\mathcal{Q} | T^\mathbb{M}\cup\{n\}\subset T, T\in\mathbb{T}_\mathcal{Q}\}} U_{T^\mathbb{M}}$ is the partial MAP solution tree in $\mathbb{T}^\mathbb{M}_\mathcal{Q}$ with the highest upper bound among those in $\mathbb{T}^\mathbb{M}_\mathcal{Q}$ that contain or extend to n.

In lieu of (7), we can derive the following proposition that implies U_n quantifies the contribution n to the upper bound of $T^\mathbb{M}_\mathcal{Q}(n)$:

Proposition 2.

$$U_{T^\mathbb{M}_\mathcal{Q}(n)} = U_n + \sum_{\{T\in\mathbb{T}_\mathcal{Q} | T^\mathbb{M}_\mathcal{Q}(n)\subset T, n\notin T\}} U_T \tag{8}$$

Moreover, it is intuitively clear that the global upper bound U is determined by the current most "promising" partial MAP solution tree, $T^\mathbb{M}_\mathcal{Q} = \operatorname{argmax}_{T^\mathbb{M}\in\mathbb{T}^\mathbb{M}_\mathcal{Q}} U_{T^\mathbb{M}}$:

Proposition 3.

$$U = \max_{T^\mathbb{M}\in\mathbb{T}^\mathbb{M}_\mathcal{Q}} U_{T^\mathbb{M}} \tag{9}$$

From Proposition 2 and 3, we see that among all the frontier nodes, only those contained in or reachable by $T^\mathbb{M}_\mathcal{Q}$ eventually contribute to U, which is very different from the pure summation case where all the frontier nodes contribute to the global bound (Lou, Dechter, and Ihler 2017). This group of frontier nodes, denoted *OPEN*$(T^\mathbb{M}_\mathcal{Q})$, can be expressed explicitly as:

$$\textit{OPEN}(T^\mathbb{M}_\mathcal{Q}) = \{n \in \textit{OPEN} \mid T^\mathbb{M}_\mathcal{Q} \cup \{n\} \subset T, T \in \mathbb{T}_\mathcal{Q}\} \tag{10}$$

It is obvious that $T^\mathbb{M}_\mathcal{Q} = T^\mathbb{M}_\mathcal{Q}(n)$ for any $n \in$ *OPEN*$(T^\mathbb{M}_\mathcal{Q})$. Thus, to quickly decrease U, we should expand $n^\star = \operatorname{argmax}_{n\in\textit{OPEN}(T^\mathbb{M}_\mathcal{Q})} U_n$, the node that contributes most to U.

Double-priority system. The above discussion also suggests a double-priority system that helps to identify $n^\star$ in each iteration. For any $n \in$ *OPEN*, a primary priority defined to be $U_{T^\mathbb{M}_\mathcal{Q}(n)}$ indicates the potential of $T^\mathbb{M}_\mathcal{Q}(n)$ to be the one that sets U; a secondary priority given by U_n quantifies the contribution of n to $U_{T^\mathbb{M}_\mathcal{Q}(n)}$. Note that these two priorities are equal for any MAX node.

However, tracking the highest priority node can be difficult in AND/OR search. In particular, both the primary and secondary priorities are *non-static*: after expanding a node in *OPEN*, the priority of other nodes in *OPEN* may change their values and relative orders. A similar effect occurs in the pure summation case (Lou, Dechter, and Ihler 2017). The double-priority system does preserve some local order invariance:

Proposition 4. For any internal node $n \in \mathcal{Q}$, expansion of any node in *OPEN*$\backslash$*OPEN*(n) will not change the relative order of nodes in *OPEN*(n) for either the primary or secondary priority.

The above local order-preserving property allows us to design an implicit priority queue for nodes in *OPEN*. To be more specific, first, for any $s \in$ *OPEN*, we define $U^\star_s = u_s$; then, for each internal node n, we maintain several quantities during search:

$$c^\star = \begin{cases} \operatorname{argmax}\limits_{c\in ch(n)} U^\star_c / u_c, & \text{if AND node } n \\ \operatorname{argmax}\limits_{c\in ch(n)} (w_c u_c, w_c U^\star_c), & \text{if OR-MAX node } n \\ \operatorname{argmax}\limits_{c\in ch(n)} w_c U^\star_c, & \text{if OR-SUM node } n \end{cases} \tag{11}$$

$$U^\star_n = \begin{cases} U^\star_{c^\star} u_n / u_{c^\star}, & \text{if AND node } n \\ w_{c^\star} U^\star_{c^\star}, & \text{if OR node } n \end{cases} \tag{12}$$

Algorithm 1 Anytime UBFS for MMAP

Initialize $\mathcal{Q} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
while termination conditions not met
  EXPANDBEST($\emptyset$) *// find best frontier, from root*
end while
function EXPANDBEST(n)
  if $n \notin$ *OPEN* *// not frontier; recurse down:*
    EXPANDBEST($c^\star$)
  else *// expand frontier node:*
    Generate children of n; $U_c^\star = u_c = h_c^+$.
    Mark any leaves as *SOLVED*.
  end if
  Update u_n via (5).
  Find $c^\star$ and update $U_n^\star$ via (11)-(12).
  if all children $c \in ch(n)$ are *SOLVED*
   or n is an OR-MAX node and $c^\star$ is *SOLVED*
    Remove $ch(n)$ from $\mathcal{Q}$; add n to *SOLVED*.
  end if
end function

In the above, "argmax" over pairs "$(,)$" means that we compute the argmax over the first component and use the second component to break ties; this reflects the primary and secondary priority structure. It is still possible for two frontier nodes to have the same priority: if those two nodes have the same node type, e.g., both are MAX nodes, we break ties randomly for simplicity; if they have different node types, we favor the MAX node over the SUM node because we prefer to reach a full MAP solution tree as early as possible.

The overall algorithm is present in Alg. 1. Note that the current best partial MAP configuration $x_{T_{\mathcal{Q}}^{\text{M}}}$ serves as an anytime approximate, and can be extended into a full MAP configuration in some greedy way if required.

Proposition 5. In each iteration, Alg. 1 finds a top-priority frontier node to expand.

See Appendix for proof.

Proposition 6. The time complexity of each node expansion and update in Alg. 1 is bounded by $O(h(b+d))$ where h is the pseudo tree height, b is the max branching factor of the pseudo tree, and d is the max variable domain size.

Memory-limited UBFS

As memory usage can quickly become a major bottleneck for a best-first search algorithm, we apply a variant of SMA* (Russell 1992) similar to that in Lou, Dechter, and Ihler (2017), so that near the memory limit, we continue expanding nodes in a best-first way, but remove low-priority nodes from $\mathcal{Q}$, in such a way that they will be re-generated once the high-priority subtrees are tightened or solved. We simply modify our updates in two ways: (1) at each node n, we also track the lowest-priority *removable* descendant of n; and (2) we force $(u_n, U_n^\star)$ to be updated monotonically, to avoid worsening the bounds or overestimating the priority when subtrees are removed and later re-generated. The resulting memory-limited best-first algorithm is shown in Alg. 2.

Algorithm 2 Memory-limited anytime UBFS for MMAP

Initialize $\mathcal{Q} \leftarrow \{\emptyset\}$ with the root $\emptyset$.
while termination conditions not met
  if memory OK: $n \leftarrow$EXPANDBEST($\emptyset$)
  else $n \leftarrow$REMOVEWORST($\emptyset$)
  end if
end while
function REMOVEWORST(n)
  if $ch(n) \subset$ *OPEN* *// worst removable node*
    Remove $ch(n)$ from $\mathcal{Q}$; mark n in *OPEN*.
  else *// or recurse toward worst*
    REMOVEWORST(c^-)
  end if
  Update c^-, u_n^- and U_n^- via (13)-(15).
end function
function EXPANDBEST(n)
  // As in Alg. 1, except:
  Ensure $(u_n, U_n^\star)$ updated monotonically
  Update c^-, u_n^- and U_n^- via (13)-(15).
end function

For convenience, we define a node as removable if its children are all in *OPEN*, and "remove" it by deleting its children and re-adding it to *OPEN*; this simplifies tracking and re-expanding removed nodes. To do so, we introduce two quantities u_n^- and U_n^- that are defined for internal nodes of $\mathcal{Q}$. For each n that is removable, we set $U_n^- = U_n^\star$, $u_n^- = u_n$ if n is a MAX node and $u_n^- = U_n^\star$ if n is a SUM node. The bottom-up recursion is as follows:

$$c^- = \begin{cases} \underset{c \in rm(n)}{\text{argmin}}(u_c^-/u_c, U_c^-/u_c), & \text{if AND node } n \\ \underset{c \in rm(n)}{\text{argmin}}(w_c u_c^-, w_c U_c^-), & \text{if OR node } n \end{cases} \tag{13}$$

$$u_n^- = \begin{cases} u_n, & \text{if AND-MAX } n \text{ \& OR-SUM } c^- \\ u_{c^-}^- u_n / u_{c^-}, & \text{if AND-MAX } n \text{ \& OR-MAX } c^- \\ w_{c^-} u_{c^-}^-, & \text{if OR node } n \end{cases} \tag{14}$$

$$U_n^- = \begin{cases} U_{c^-}^- u_n / u_{c^-}, & \text{if AND node } n \\ w_{c^-} U_{c^-}^-, & \text{if OR node } n \end{cases} \tag{15}$$

where $rm(n) = ch(n) \backslash$ *OPEN*, i.e., the children of n not in *OPEN*. Then, to remove a node, we search downward along the worst children c^-, and remove n when its children are all in *OPEN*.

Empirical Evaluation

We evaluate the anytime performance of our proposed algorithm (Alg. 2, called UBFS) against three baselines which can provide anytime bounds on three benchmarks. The baselines include AAOBF (Marinescu et al. 2017), a state-of-the-art search algorithm, XOR_MMAP (Xue et al. 2016), a recent random hashing based approach, and AFSE (Mauá and de Campos 2012), a factor-set elimination scheme. AAOBF and AFSE can provide anytime deterministic bounds, while XOR_MMAP provides anytime stochastic bounds. Generally,

Table 1: Statistics of the three evaluated benchmark sets. "avg. induced width" and "avg. pseudotree depth" are computed given 50% of variables randomly selected as MAX variables.

	grid	promedas	protein
# instances	100	100	50
avg. # variables	764.48	1063.84	99.96
avg. # of factors	764.48	1076.84	355.84
avg. max domain size	2.00	2.00	77.94
avg. max scope	3.00	3.00	2.00
avg. induced width	190.59	136.72	35.28
avg. pseudotree depth	218.23	170.50	43.72

the baselines are chosen to cover recent anytime bounding schemes from different categories.

The benchmark set includes three problem domains: grid networks (grid), medical diagnosis expert systems (promedas), and protein, made from the "small" protein side-chains of Yanover and Weiss (2002). grid is a subset of the grid dataset used in Marinescu et al. (2017) with "small" instances (# variables less than 324) excluded because they can be solved exactly during heuristic construction and so are less interesting. promedas is the same dataset as that used in Marinescu et al. (2017). We tested two settings of MAX variables. In the first setting, 50% of the variables are randomly selected as MAX variables, which is the same setting used in Marinescu et al. (2017). To compare those algorithms on instances with relatively hard internal summation problems, we decrease the percentage of MAX variables to 10% for the second setting. The statistics of the benchmarks in Table 1 show these instances are very challenging.

The time budget is set to 1 hour for all experiments. We allot 4GB memory to all algorithms, with 1GB extra memory to AAOBF for caching. For our experiments, we use the weighted mini-bucket (Liu and Ihler 2011) heuristics, whose memory usage is roughly controlled by an *ibound* parameter. For a given memory budget, we first compute the largest ibound that fits in memory, then use the remaining memory for search. Since AAOBF also uses weighted mini-bucket heuristics, the same ibound is shared during heuristic construction between our proposed algorithm and AAOBF. Implementations of all methods are in C/C++ by the original authors except AFSE, implemented by the authors of Marinescu et al. (2017). The step size used by AFSE to control the partitioning of the factor sets is set to 1 since litte difference in performance was observed for larger values (Marinescu et al. 2017). For XOR_MMAP, we adopt parameter values suggested by its authors. Unfortunately, XOR_MMAP failed to produce valid bounds on many of our instances in the allotted time, perhaps because it maintains multiple copies of the problem instance and must solve internal NP-hard parity constraint problems. Despite its good solutions and bounds on small instances (Xue et al. 2016), it does not appear to scale well to our harder benchmarks.

Individual Results

Our algorithm is designed to improve upper bounds in an anytime fashion. Fig. 2 shows the methods' anytime behavior on individual instances from each benchmark. The lower bounds from UBFS in those plots, corresponding to the best solutions found so far by UBFS at the respective timestamps, are computed offline and are intended only for reference, to give a sense of the quality of the anytime MAP configurations predicted by UBFS.

Table 2: Number of instances with non-trivial bounds at three times (1 min, 10 min, and 1 hour resp.) for each benchmark. 50% MAX variables. The highest for each setting is bolded.

	grid	promedas	protein
# instances	100	100	50
Timestamp: 1min/10min/1hr			
UBFS	**100/100/100**	**100/100/100**	**46/50/50**
AAOBF	98/**100/100**	**100/100/100**	23/31/34
AFSE	0/0/0	25/27/27	7/7/7

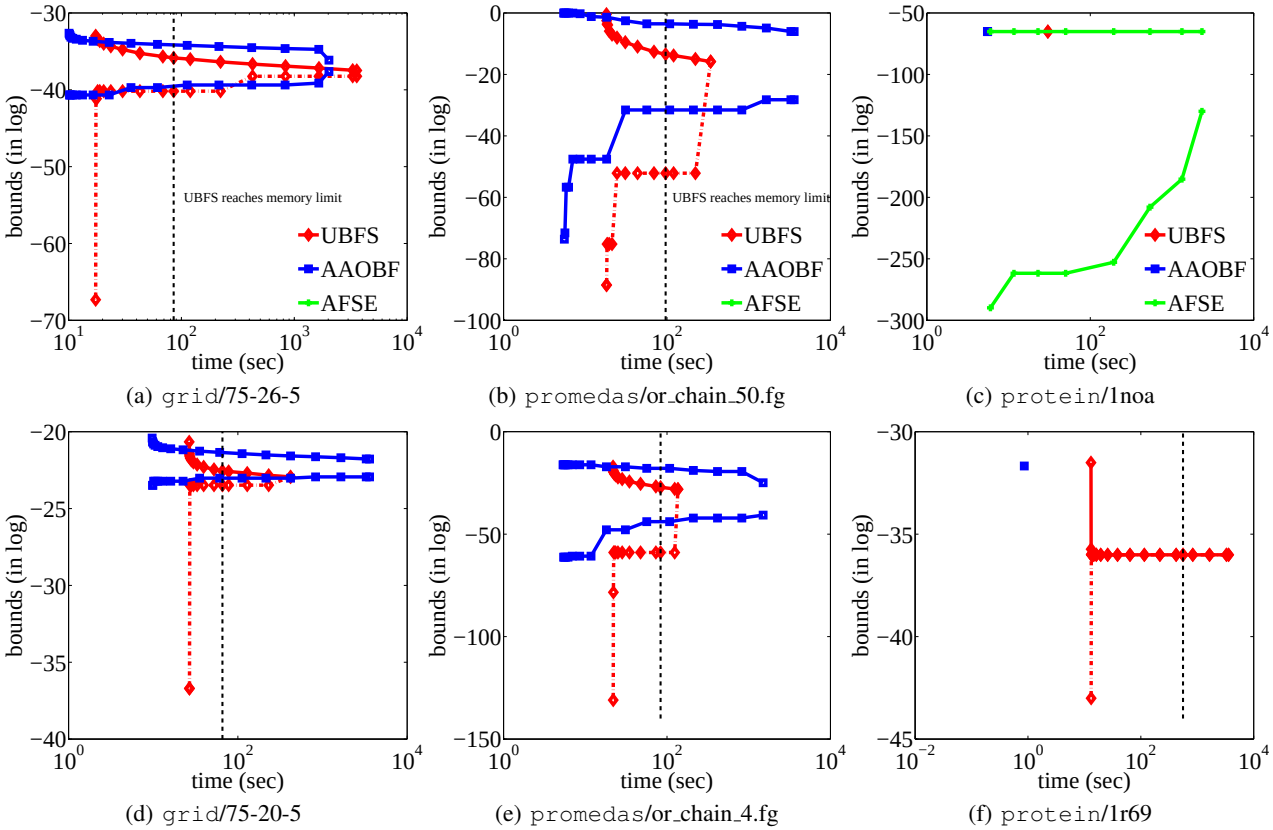
From Fig. 2, we observe that only UBFS and AAOBFS are able to provide valid (upper) bounds on all the 6 instances. AFSE runs out of memory on all but one instance before providing any bounds, which was fairly typical; for comparison, UBFS and AAOBF are able to solve this instance optimally. We commonly observed that when AFSE is able to produce bounds, UBFS and AAOBF usually produce better bounds or even find optimal solutions. XOR_MMAP failed to provide any solutions or bounds for these instances.

When UBFS reaches the memory limit, it keeps improving the upper bounds and optimally solves some of the problems (e.g., Fig. 2(b),2(d),2(e)). As expected, memory is a bottleneck and UBFS usually reaches the 4GB limit (vertical lines in the plots) within a few minutes of search, but continues to improve its bound by pruning low-priority nodes. In contrast, AAOBF terminates when memory is used up (e.g., Fig. 2(a)). Note that UBFS typically runs into the memory limit faster than AAOBF; in Fig. 2(a) UBFS uses up its memory in 100 seconds, while AAOBF reaches the limit after 1000 seconds. This is because UBFS is a pure best-first scheme, while AAOBF is a hybrid scheme that solves internal summation problems using depth-first search; if the DFS step is slow (hard summation problems), the best-first portion of the search will proceed slowly. Fig. 2(f) shows an example where this summation is so slow that AAOBF fails to provide any bounds beyond its inital heuristic upper bound; in contrast, UBFS quickly finds a much higher-quality upper bound (albeit without a corresponding online lower bound).

Collective Results

We evaluate some statistics across algorithms to compare their performance on the benchmarks. Since XOR_MMAP generally fails to provide bounds, we exclude it in the sequel.

Responsiveness. Responsiveness characterizes how fast an algorithm produces non-trivial bounds. For UBFS and AAOBF, we require them to produce bounds other than the initial heuristic bound. From Table 2, we can see that UBFS responds quickly on almost all the instances within 1 minute, except for 4 protein instances that require more time to process. AAOBF is responsive on most grid and promedas instances, but performs worse on a number of



(a) grid/75-26-5 (b) promedas/or_chain_50.fg (c) protein/1noa

(d) grid/75-20-5 (e) promedas/or_chain_4.fg (f) protein/1r69

Figure 2: Anytime bounds for two instances per benchmark, with 50% MAX variables. AFSE is missing if it ran out of memory before producing bounds; XOR_MMAP failed to produce bounds on these instances. UBFS lower bounds are computed offline and shown only for reference. Black dotted lines mark UBFS reaching the 4GB memory limit; time budget 1 hour.

Table 3: Number of instances that an algorithm achieves the best upper bounds at each timestamp (1 min, 10 min, and 1 hour) for each benchmark. 50% MAX variables. The best for each setting is bolded.

	grid	promedas	protein
# instances	100	100	50
Timestamp: 1min/10min/1hr			
UBFS	**85/84/89**	**83/86/87**	**46/50/50**
AAOBF	33/46/44	47/47/47	17/16/18
AFSE	0/0/0	0/0/0	5/5/5

Table 4: Number of instances that an algorithm achieves best upper bounds at each given timestamp (1 min, 10 min, and 1 hour) for each benchmark. 10% MAX variables. The best for each setting is bolded.

	grid	promedas	protein
# instances	100	100	50
Timestamp: 1min/10min/1hr			
UBFS	**99/100/100**	**88/99/99**	**43/50/50**
AAOBF	1/1/3	17/12/17	15/9/9
AFSE	0/0/0	10/8/10	7/7/7

protein instances due to their very hard internal summation problems. AFSE is not competitive and fails to produce any bounds on the grid instances.

Bound quality. We compare anytime upper bounds among the algorithms. Table 3 shows the number of instances for which an algorithm achieves the tightest upper bounds at each of three timestamp. From Table 3, we see that our algorithm does best on this task across all benchmark/timestamp settings, by a large margin. Again, the advantage is most significant on the protein instances, and again, AAOBFS performs better than AFSE.

Harder summation. UBFS avoids some unnecessary evaluation of the internal summation problems, thus it is expected to be suitable for those MMAP problems with hard internal summation problems. To assess this setting, we decrease the percentage of MAX variables from 50% to 10%. Table 4 shows the relative upper bound quality. By comparing Table 3 and Table 4, we can see that UBFS gains a larger advantage when the search space of MAX variables is smaller, while the summation tasks are relatively harder.

Conclusion

In this paper, we present an anytime anyspace AND/OR best-first search algorithm to improve upper bounds for MMAP

problems. We cast max- and sum- inference into one best-first search framework. The specially designed double-priority system allows our algorithm to identify the current most promising partial MAP solution tree and expand the frontier node that contributes most to the upper bound of that partial MAP solution tree, which often leads to quick reduction on the global upper bound. Our algorithm avoids some unnecessary exact computation of internal conditional summation problems and thus has a significant advantage especially on those instances with hard internal summation problems. Empirical results demonstrate that our approach with heuristics extracted from weighted mini-bucket is superior to those state-of-the-art baselines on various settings.

Acknowledgements

We thank all the reviewers for their helpful feedback. We appreciate the help from William Lam, Junkyu Lee, Radu Marinescu, Wei Ping, and Yexiang Xue.

This work is sponsored in part by NSF grants IIS-1526842, IIS-1254071, and by the United States Air Force under Contract No. FA8750-14-C-0011 and FA9453-16-C-0508.

References

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Dechter, R., and Mateescu, R. 2007. And/or search spaces for graphical models. *Artif. Intell.* 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.; Geffner, H.; and Halpern, J. Y. 2010. *Heuristics, Probability and Causality. A Tribute to Judea Pearl*. College Publications.

Dechter, R. 2013. Reasoning with probabilistic and deterministic graphical models: Exact algorithms. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 7(3):1–191.

Doucet, A.; Godsill, S. J.; and Robert, C. P. 2002. Marginal maximum a posteriori estimation using markov chain monte carlo. *Statistics and Computing* 12(1):77–84.

Kiselev, I., and Poupart, P. 2014. Policy optimization by marginal-map probabilistic inference in generative models. In *Proceedings of the 13th international conference on Autonomous agents and multi-agent systems*, 1611–1612. International Foundation for Autonomous Agents and Multiagent Systems.

Lee, J.; Marinescu, R.; Dechter, R.; and Ihler, A. 2016. From exact to anytime solutions for marginal map. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 3255–3262.

Liu, Q., and Ihler, A. 2011. Bounding the partition function using Hölder's inequality. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*.

Liu, Q., and Ihler, A. 2013. Variational algorithms for marginal map. *Journal of Machine Learning Research* 14(1):3165–3200.

Lou, Q.; Dechter, R.; and Ihler, A. 2017. Anytime anyspace and/or search for bounding the partition function. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

Marinescu, R.; Lee, J.; Ihler, A.; and Dechter, R. 2017. Anytime best+ depth-first search for bounding marginal map. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

Marinescu, R.; Dechter, R.; and Ihler, A. 2014. And/or search for marginal map. In *UAI*, 563–572.

Marinescu, R.; Dechter, R.; and Ihler, A. 2015. Pushing forward marginal map with best-first search. In *IJCAI*, 696–702.

Mauá, D., and de Campos, C. 2012. Anytime marginal maximum a posteriori inference. In *ICML*.

Otten, L., and Dechter, R. 2012. Anytime and/or depth-first search for combinatorial optimization. *AI Communications* 25(3):211–227.

Park, J., and Darwiche, A. 2003. Solving map exactly using systematic search. In *Proceedings of the 19th conference on Uncertainty in Artificial Intelligence*, 459–468. Morgan Kaufmann Publishers Inc.

Park, J. 2002. Map complexity results and approximation methods. In *Proceedings of the 18th conference on Uncertainty in artificial intelligence*, 388–396. Morgan Kaufmann Publishers Inc.

Ping, W.; Liu, Q.; and Ihler, A. 2014. Marginal structured svm with hidden variables. In *Proceedings of the 31st International Conference on Machine Learning*, 190–198.

Ping, W.; Liu, Q.; and Ihler, A. 2015. Decomposition bounds for marginal map. In *Advances in Neural Information Processing Systems*, 3267–3275.

Russell, S. 1992. Efficient memory-bounded search methods. In *Proceedings of the 10th European Conference on Artificial Intelligence*, ECAI '92, 1–5.

Valiant, L. 1979. The complexity of computing the permanent. *Theoretical Computer Science* 8(2):189 – 201.

Wainwright, M., and Jordan, M. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning* 1(1-2):1–305.

Xue, Y.; Li, Z.; Ermon, S.; Gomes, C. P.; and Selman, B. 2016. Solving marginal map problems with np oracles and parity constraints. In *Advances in Neural Information Processing Systems*, 1127–1135.

Yanover, C., and Weiss, Y. 2002. Approximate inference and protein-folding. In *Advances in neural information processing systems*, 1457–1464.

Yuan, C., and Hansen, E. A. 2009. Efficient computation of jointree bounds for systematic map search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 1982–1989.

Yuan, C.; Lu, T.-C.; and Druzdzel, M. J. 2004. Annealed map. In *UAI*, 628–635. AUAI Press.

Appendix

Proof of Proposition 5

Proposition 5. In each iteration, Alg. 1 finds a top-priority frontier node to expand.

We will prove the following claims from which we can easily derive the proposition.

For any internal node $n \in \mathcal{Q}$, we claim: First, its best child $c^\star$ found via (11) is an ancestor of a top-priority descendant of n in *OPEN*.

Second,

$$U_n^\star g_n \prod_{s \in branch(n)} u_s \tag{16}$$

is the secondary priority value of n' top-priority descendant in *OPEN*.

We will prove the above claims via induction on the height of nodes in $\mathcal{Q}$, where the height for a node is defined as the distance from that node to its furthest descendant in *OPEN*. For example, a frontier node has height 0; a node has all its children in *OPEN* has height 1.

To begin with, the first claim is vacuously true for all frontier nodes, and the second claim is true for all frontier nodes by definition. Then, suppose the claims hold for those nodes of height no greater than some $h \geq 0$, we will show they also hold for nodes of height $h+1$. Now, let $n \in \mathcal{Q}$ be a node of height $h+1$; all its children have height no greater than h and thus the claims apply to them.

If n is an AND node, for any $c_1, c_2 \in ch(n)$, it is easy to see that their top-priority frontier descendants have the same primary priority. Thus, we only have to compare the secondary priority of their top-priority frontier descendants. Since

$$U_{c_1}^\star g_{c_1} \prod_{s \in branch(c_1)} u_s \tag{17}$$

is the secondary priority value of c_1's top-priority frontier descendant according to (16). By applying the facts that $g_n = g_{c_1}$ and $u_n = \prod_{c \in ch(n)} u_c$ to the above quantity, we have

$$U_{c_1}^\star / u_{c_1} u_n g_n \prod_{s \in branch(n)} u_s \tag{18}$$

Thus, (11) finds $c^\star$ that leads to a top-priority descendant of AND node n, which has the secondary priority

$$U_{c^\star}^\star / u_{c^\star} u_n g_n \prod_{s \in branch(n)} u_s \tag{19}$$

Combining the above and (12), we know the second claim is true for n as well.

If n is an OR-MAX node, for any $c_1 \in ch(n)$, it is easy to see that

$$g_{c_1} u_{c_1} \prod_{s \in branch(c_1)} u_s \tag{20}$$

is the primary priority value of c_1's top-priority frontier descendant, which can be re-written as

$$w_{c_1} u_{c_1} g_n \prod_{s \in branch(n)} u_s \tag{21}$$

by considering the facts that $g_{c_1} = w_{c_1} g_n$ and $branch(c_1) = branch(n)$. According to the second claim and $g_{c_1} = w_{c_1} g_n$, the secondary priority value of c_1's top-priority frontier descendant can be re-written as

$$w_{c_1} U_{c_1}^\star g_n \prod_{s \in branch(n)} u_s \tag{22}$$

Thus, $c^\star$ found via (11) leads to a top-priority descendant. Also, the second claim holds for n in lieu of (12).

If n is an OR-SUM node, we know all its frontier descendants share the same primary priority. The analysis on the secondary priority is the same as that of the OR-MAX case.

All in all, we can see that the two claims hold for a node of height $h+1$. By induction, we know these claims hold for all internal nodes of $\mathcal{Q}$, which implies Proposition 5.