Journal of Artificial Intelligence Research 63 (2018) 875-921 Submitted 06/18; published 12/18

AND/OR Search for Marginal MAP

Radu Marinescu RADU.MARINESCU@IE.IBM.COM
IBM Research
Dublin, Ireland
Junkyu Lee JUNKYUL@ICS.UCI.EDU
Rina Dechter DECHTER@ICS.UCI.EDU
Alexander Ihler IHLER@ICS.UCI.EDU
University of California, Irvine
Irvine, CA 92697, USA

Abstract

Mixed inference such as the marginal MAP query (some variables marginalized by summation and others by maximization) is key to many prediction and decision models. It is known to be extremely hard; the problem is NP^{PP}-complete while the decision problem for MAP is only NP-complete and the summation problem is #P-complete. Consequently, approximation anytime schemes are essential. In this paper, we show that the framework of heuristic AND/OR search, which exploits conditional independence in the graphical model, coupled with variational-based mini-bucket heuristics can be extended to this task and yield powerful state-of-the-art schemes. Specifically, we explore the complementary properties of best-first search for reducing the number of conditional sums and providing time-improving upper bounds, with depth-first search for rapidly generating and improving solutions and lower bounds. We show empirically that a class of solvers that interleaves depth-first with best-first schemes emerges as the most competitive anytime scheme.

1. Introduction

Probabilistic graphical models provide a powerful framework for reasoning about conditional dependency structures over many variables. For such models, the *Marginal MAP* (MMAP) inference query is a particularly difficult yet important task, corresponding to finding the most probable configuration (or maximizing the probability) over a subset of variables, called MAP variables, after marginalizing (summing over) the remainder of the model. MMAP inference arises in many situations, especially in diagnosis and planning tasks, in which the most natural specification of the model contains many variables whose values we do not care about predicting, but which create interdependence among the variables of interest. For example, in diagnosis tasks, given observations, we seek to optimize over a subset of diagnosis variables representing potentially failing components in a system (de Kleer, Mackworth, & Reiter, 1990; Geffner & Bonet, 2013). Furthermore, a probabilistic conformant planning task which finds a sequence of actions that maximizes the probability of reaching a goal state without observations can be solved by MMAP inference (Lee, Marinescu, & Dechter, 2014). More generally, the optimal policy of a finite horizon probabilistic planning task can be found by MMAP inference due to the equivalence between MMAP inference and the maximum expected utility computation (Mauá, 2016).

One reason marginal MAP queries are difficult is that the max and sum operations do not commute, forcing variable elimination to be processed along a "constrained" order that eliminates summation variables first, and often has significantly higher induced width than the graph itself

(Dechter, 1999, 2013). In general, MMAP is NP^{PP}-complete and it can be NP-hard even on tree structured graphs (Park & Darwiche, 2004), while the decision problem for MAP is only NP-complete (Shimony & Charniak, 1991) and the summation problem is #P-complete (Cooper, 1990).

The difficulty of solving marginal MAP exactly motivates *anytime* algorithms, which provide a confidence interval on the exact solution that is gradually improved over time. Similarly, it is important that any practical algorithm be *anyspace*, in the sense that it can continue to make progress once the available memory is exhausted. In this paper, we focus on the framework of heuristic search to provide these properties.

In particular, we explore several major ways in which heuristic search for marginal MAP can be made faster and more effective: using best-first techniques, exploiting AND/OR decomposition structure, and making improvements to the guiding heuristic function.

Past algorithms for solving marginal MAP have been typically based on depth-first branch and bound search, e.g., Park and Darwiche (2003), Yuan and Hansen (2009). While this approach is quick to produce an initial MAP configuration, it can be slow to improve. Evaluating or comparing any two MAP configurations involves a combinatorial sum, and when this is computationally costly, depth-first methods (which often visit a large number of such configurations) can pay a heavy price. In contrast, best-first search methods will solve far fewer summation problems to prove optimality, but will be correspondingly much slower to produce an initial solution.

The AND/OR structure provides additional improvements. AND/OR search spaces leverage conditional independence relations during search, in order to reduce the effective search space size. Although naïvely search is exponential in the number of variables, AND/OR search is exponential only in the depth of the model's *pseudo-tree* (Dechter & Mateescu, 2007; Freuder & Quinn, 1985), which may be far smaller. Moreover, if memory is utilized it can be exponential in the induced width only.

Finally, the heuristic functions used in previous work (Park & Darwiche, 2003; Yuan & Hansen, 2009) work by exactly solving a relaxed inference problem that follows an "unconstrained" order on the graph, which can have a smaller induced width than the exact marginal MAP task. However, this framework has two major drawbacks. First, many marginal MAP problems are not easily solved even with unconstrained orderings, limiting the methods' application in practice. We provide approximate extensions of Park and Darwiche (2003) and Yuan and Hansen (2009) that allow them to be applied more generally. More importantly, however, since the search process must follow the constrained order, these heuristics are by necessity *dynamic*, i.e., evaluating them at some nodes requires recomputing at least part of the variable elimination procedure, with a nontrivial computational cost. We show that a heuristic based on partition-based weighted mini-bucket (Dechter & Rish, 2003; Liu & Ihler, 2011; Marinescu, Dechter, & Ihler, 2014) is much more effective for search, in part because it allows the heuristic function to be completely pre-compiled, leading to far faster per-node evaluations.

We develop AND/OR variants of both depth- and best-first search for marginal MAP. Then, motivated by our desire for good anytime behavior and the high cost of evaluating a MAP configuration, we explore two approaches that balance the benefits of depth-first and best-first behavior: *weighted* best-first search (Pohl, 1970; Flerova, Marinescu, & Dechter, 2017), and two hybrid schemes that interleave best-first and depth-first operations. These algorithms demonstrate significantly better anytime performance than previous approaches.

We show that indeed heuristic search over AND/OR graphs, guided by such variational-based partition heuristic can advance exact solvers and more importantly, it results in anytime algorithms

Algorithm	Limited Memory	Strategy	Anytime	Upper Bounds	Lower Bounds
AOBB	Yes	Depth-first	Yes	No	Yes
AOBF	No	Best-first	No	Yes	No
RBFAOO	Yes	Recursive best-first	No	No	No
BRAOBB	Yes	Depth-first	Yes	No	Yes
WAOBF	No	Weighted best-first	Yes	No	Yes
WAOBF-REP	No	Weighted best-first	Yes	No	Yes
WRBFAOO	Yes	Weighted best-first	Yes	No	Yes
AAOBF	No	Best+depth-first	Yes	Yes	Yes
LAOBF	No	Best+depth-first	Yes	Yes	Yes

Table 1: New AND/OR search algorithms for marginal MAP.

for generating solutions and bounds that improve with time. Our empirical evaluations conclude that a class of solvers that interleave depth-first with best-first schemes emerges as most competitive anytime scheme.

In Table 1 we summarize the main contributions of this paper by listing the proposed AND/OR search algorithms together with their most important characteristics such as whether or not they can operate within restricted memory, search strategy, anytime behavior, as well as the ability to produce upper and lower bounds on the optimal marginal MAP value. Specifically, the first three algorithms (AOBB, AOBF and RBFAOO) denote the exact depth-first and best-first search schemes for computing optimal solutions. The subsequent four algorithms (BRAOBB, WAOBF, WAOBF-REP and WRAOBF) are the anytime schemes based on either depth-first or weighted best-first search that produce improved quality solutions (corresponding to lower bounds) in an anytime fashion. Finally, the last two algorithm (LAOBF and AAOBF) correspond to the hybrid depth+best-first search bounding schemes that compute not only anytime lower bounds but also anytime upper bounds on the optimal marginal MAP value. We will show empirically: (1) the effectiveness of the AND/OR search algorithms as exact schemes against previous unconstrained join-tree based methods exploring OR search spaces and (2) our new best+depth-first search approach produces superior quality solutions more quickly than the pure best-first or depth-first search as well as the weighted schemes.

The paper is structured as follows. In Section 2 we give notation and background on graphical models and marginal MAP. Section 3 describes some heuristic functions for marginal MAP. Then, Section 4 describes depth-first and best-first search, develops AND/OR versions for marginal MAP, and evaluates and analyzes their performance characteristics as exact solvers. Section 5 develops two families of anytime search that blend the positive qualities of depth- and best-first methods, based on using weighted best-first or interleaving best- and depth-first operations, respectively. We assess the performance of these two frameworks on a number of benchmark problem classes. Finally, we describe some connections to related work in Section 6, and discuss our conclusions in Section 7. The paper is a significantly expanded and unified presentation of a line of work by the authors, including (Marinescu et al., 2014; Marinescu, Dechter, & Ihler, 2015; Lee, Marinescu, Dechter, & Ihler, 2016; Marinescu, Lee, Dechter, & Ihler, 2017).

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, ie, $\mathbf{X}_\alpha = var(\psi_\alpha) = \{X_i : i \in \alpha\}$. The function scopes imply a *primal graph* whose vertices are the variables and which includes an edge connecting 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

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

The *partition function*, Z, normalizes the probability to sum to one.

Marginal MAP problems distinguish maximization variables (called MAP variables) and summation variables (the others). Let $\mathbf{X}_S$ be the subset of summation variables in $\mathbf{X}$ and $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$ be the complement of $\mathbf{X}_S$. The *Marginal MAP* problem is to find the assignment x^*_M to variables $\mathbf{X}_M$ that maximizes the value of the marginal distribution after summing out variables $\mathbf{X}_S$:

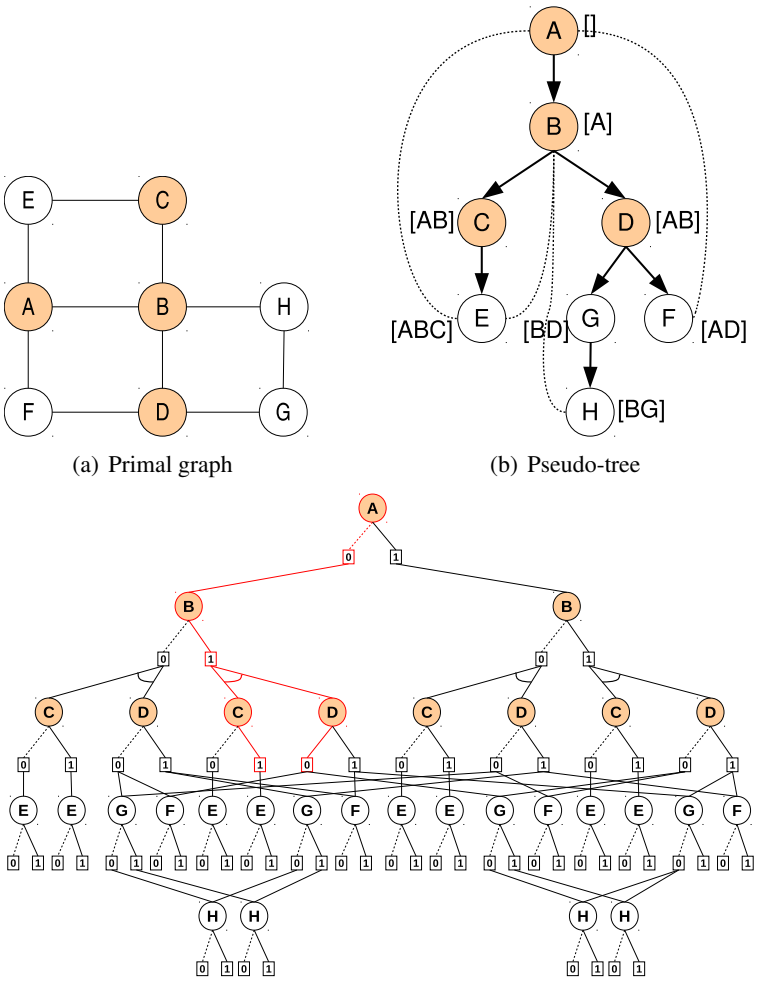
$$x^*_M = \arg\max_{\mathbf{X}_M} \sum_{\mathbf{X}_S} \prod_{\alpha \in F} \psi_\alpha(x_\alpha) \tag{1}$$

Example 1. *Figure 1(a) depicts the primal graph of an undirected graphical model (called also a Markov network) representing a distribution over 8 variables, $\mathbf{X} = A, \ldots, H$ with 8 functions defined by the arcs (each pair is a scope of one function). The highlighted variables $\{A, B, C, D\}$ denotes the MAP variables, in a possible marginal MAP task.*

2.1 Bucket Elimination

An exact solution to (1) can be obtained by using the *bucket elimination* (BE) algorithm, which eliminates (sums or maximizes over) the variables in sequence (Dechter, 1999). Given an ordering of the variables in which all the max variables come before the sum variables, BE partitions the functions into buckets, each associated with a single variable. A function is placed in the bucket of its argument that appears latest in the ordering. BE processes each bucket, from last to first, by multiplying all functions in the current bucket and eliminating the bucket's variable (by summation for sum variables and by maximization for MAP variables), resulting in a new function which is then placed in an earlier bucket, depending on its scope. Since the size of this function is exponential in its number of arguments, the complexity of BE is time and space exponential in the *induced width* of the primal graph along the elimination order (Dechter, 1999).

Because the order in a marginal MAP task is constrained to place all max variables before all sum variables, we refer to it as a *constrained elimination order*, and to its induced width as the *constrained induced width*, denoted w^*_c. The constrained induced width is always higher than the induced width of the best unconstrained order, sometimes significantly so. Marginal MAP is harder than pure optimization (MAP) tasks because evaluating the cost of the MAP assignment requires solving a summation over the sum variables, and harder than pure summation because it may force an undesirable (constrained) elimination order. For example, while summation is efficient in tree-structured graphs (induced width 1), it is easy to specify MAP variables such that the resulting



(a) Primal graph (b) Pseudo-tree

(c) AND/OR search graph

Figure 1: Example (a) primal graph of a graphical model over eight variables; (b) a pseudo-tree corresponding to the ordering $d = (A, B, C, D, E, G, F, H)$; (c) the resulting AND/OR search graph. An example solution tree is indicated in red (see Section 4.2).

marginal MAP problem is NP-hard. This effect means that many marginal MAP problems are impractical to solve exactly through bucket elimination, necessitating approximations and search methods.

2.2 AND/OR Search Spaces

AND/OR search is guided by a spanning *pseudo-tree* of the graphical model's primal graph (in which any arc of the model not in the tree is a back-arc in the pseudo-tree) and the search space is exponential in the depth of the pseudo-tree (rather than in the number of variables). Therefore, any of the inference tasks over a graphical model can be computed by traversing the AND/OR search tree (Dechter & Mateescu, 2007) even with linear memory. Moreover, when memory can be used the

AND/OR search space can be compacted into a graph whose size is exponential in the induced width only.

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 included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in $\mathcal{T}$ to one of its ancestors. The arcs in E' may not all be included in E.*

Figure 1(b) shows a pseudo-tree of depth 5 of the graphical model in Figure 1(a) that could be generated by a depth-first traversal of the graph in Figure 1(a) (for more details on how to construct pseudo-trees, see Dechter and Mateescu (2007), Marinescu and Dechter (2009a)). 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 weighted according to $\mathbf{F}$. The AND/OR search space can be compacted further by merging nodes that root identical subtrees. One way to identify such nodes is by using the notion of *context* (Dechter & Mateescu, 2007).

DEFINITION **2** (parents, context)**.** *Given a graphical model $\mathcal{M}$ with primal graph G and pseudo-tree $\mathcal{T}$, the* parents *of variable X_i, denoted by pa_i or pa_{X_i}, are the ancestors of X_i in $\mathcal{T}$ that have connections (in G) to X_i or to descendants of X_i in $\mathcal{T}$. Let n be an OR node labeled by variable X_i in the AND/OR search tree $S_{\mathcal{T}}$. A variable assignment to pa_i defines the* context *of node n.*

It was shown that any two OR nodes n_1 and n_2 with identical contexts root identical subproblems in $S_{\mathcal{T}}$ and therefore can be merged (Dechter & Mateescu, 2007). Merging all nodes that have identical contexts yields the *context-minimal AND/OR graph* shown in Figure 1(c) (the parents sets of the variables are shown next to each node of the pseudo-tree in Figure 1(b)). The size of this search space can be bounded exponentially by the treewidth w^* (or, the induced width). Searching the AND/OR context-minimal graph can be bounded to be time and space exponential in the treewidth w^*, the same complexity bound obeyed by message-passing schemes such as bucket elimination (Dechter, 1999). Yet, search schemes can work with more flexible representations which is not necessarily table-based, and invite methods that can trade memory for time and lead smoothly into anytime schemes.

3. Improved Partitioning Based Upper Bounds for Marginal MAP

Informed search algorithms rely heavily on a heuristic function to guide the search process, and bound the quality of potential solutions. In practice, these heuristics are usually constructed by approximating the exact bucket elimination process described in Section 2. In this section, we describe two common frameworks for heuristic construction: *mini-bucket* (Dechter & Rish, 2003) and *reordered join-tree* (Park & Darwiche, 2003) heuristics. Both frameworks have drawbacks: in particular, while the reordered heuristic has been shown to outperform mini-bucket, its computational complexity is not easily controlled, and by construction it can require significant recomputation at some nodes of the search. We fix the former issue using a heuristic called *mini-cluster-tree*, which further approximates the reordered join-tree in a computationally bounded way. In order to develop a more efficient heuristic, we revisit mini-bucket and augment it with several recent developments to improve its quality on pure summation (Liu & Ihler, 2011) and pure MAP (Ihler, Flerova, Dechter, & Otten, 2012), giving a *weighted mini-bucket* heuristic for marginal MAP that can be used in AND/OR search. This provides a stronger heuristic than standard mini-bucket, while remaining highly efficient to evaluate at each node.

3.1 Mini-Bucket Elimination

Mini-Bucket Elimination (MBE) is a classic relaxation of the exact variable elimination problem in (1) by approximating each elimination operator (max or sum) to enable the user to control a bound on the space and time complexity. Following the exact bucket elimination procedure, each bucket is further partitioned into smaller subsets, called *mini-buckets*, each containing at most i distinct variables (where i is a user-selected parameter called the i-bound). The mini-buckets are processed independently, resulting in messages over fewer variables and thus require less time and memory. MBE processes the sum buckets and max buckets differently. Max mini-buckets (of variables in $\mathbf{X}_M$) are eliminated by maximization, while for variables in $\mathbf{X}_S$, one (arbitrarily selected) mini-bucket is eliminated by summation, while the rest of the mini-buckets are eliminated by maximization. The complexity of the algorithm is time and space exponential only in its i-bound, which can be selected to enforce available computational resources. When i is sufficiently large (i.e., $i > w_c^*$), MBE coincides with full bucket elimination, yielding the exact answer.

By design, these mini-bucket messages upper bound the exact elimination process, giving an upper bound on the optimal marginal MAP value. Moreover, given any partial configuration that respects this ordering (i.e., consists of values for the last k eliminated variables), we can evaluate an upper bound on the conditional problem from the messages computed during the approximate elimination. These bounds can be used as a search heuristic, and are very efficient to evaluate, requiring at most a few table lookup operations per search node. MBE(i) has regularly been used as an effective heuristic for depth-first branch and bound search on MAP problems (Marinescu & Dechter, 2009a, 2009b).

3.2 Unconstrained-order, Join-tree Based Upper Bounds

An alternative approach for upper bounding marginal MAP was proposed by Park and Darwiche (2003). In this method, a join-tree is constructed along an *unconstrained* elimination order that interleaves the MAP and the sum variables. Since reversing the order of the sum and max elimination in (1) gives an upper bound on its value, a standard two-pass evaluation of the join-tree for this reordered problem is guaranteed to generate an upper bound on the optimal marginal MAP value. The complexity of the evaluation is exponential in the unconstrained induced width of the underlying primal graph, which can be less than the constrained induced width. When this computational cost is not too high, this method was shown to be more effective than MBE (Park & Darwiche, 2003).

Unfortunately, there are two significant drawbacks to this approach. The first is that the computational requirements are not completely under user control. In many models, even the unconstrained induced width remains prohibitively high, which limits the method's practical application. In order to use these techniques in general models, we may need to further approximate them; in the sequel, we develop a mini-cluster tree elimination heuristic to overcome this issue.

The second drawback of unconstrained order methods is that they are by definition *dynamic*, meaning that, at some search nodes, they may require some significantly non-trivial computation to evaluate the heuristic bound. Yuan and Hansen (2009) improve on (Park & Darwiche, 2003), reducing the number of such re-evaluations by carefully caching and reusing messages, but still require inference re-computation at some nodes. Fundamentally, this is a consequence of the fact that the search must proceed over the max variables; whenever a re-ordered max variable is conditioned on in the search, the join tree messages no longer provide a usable heuristic and must be re-computed.

For this reason, we also propose an improved form of (constrained-order) mini-bucket heuristic for marginal MAP.

3.3 Mini-Cluster-Tree Elimination

To control the computational costs of the unconstrained order based heuristic, we apply an approximation with bounded complexity, related to mini-bucket but with computations organized more similarly to join-tree inference, called *Mini-Cluster-Tree Elimination* (MCTE) (Dechter, Kask, & Larrosa, 2001). In MCTE, we pass messages along the structure of the join-tree, except that when computing a message, rather than combining all the functions in the cluster, we first partition it into mini-clusters, such that each mini-cluster has a bounded number of variables (the i-bound). Each mini-cluster is then processed independently to compute a set of outgoing messages. Like MBE, this procedure produces an upper bound on the results of exact inference, and increasing i typically provides tighter bounds, but at higher computational cost. In spirit, both MBE and MCTE are quite similar and both methods could be applied to either constrained or unconstrained order eliminations, but in this paper for consistency we always use MBE-based relaxations on constrained orderings, and MCTE relaxations on unconstrained orderings.

The dynamically evaluated nature of reordered (unconstrained order) heuristics, however, is more fundamental to the approach. While dynamic heuristics are often tighter than statically compiled ones, and can enable additional, useful techniques such as dynamic search orders, the efficiency of evaluating a static heuristic allows it to visit more nodes in a given time. For this reason, we explore an improved static heuristic based on weighted mini-bucket elimination, which we show experimentally leads to a more effective search process.

3.4 Weighted Mini-Bucket Elimination and Cost-shifting

Weighted Mini-Bucket (WMB) (Liu & Ihler, 2011) is a recent enhancement of the mini-bucket scheme for likelihood (summation) tasks. It replaces the naïve mini-bucket bound with Hölder's inequality. For a given variable X_k, the mini-buckets Q_{kr} associated with X_k are each assigned a non-negative *weight* $w_{kr} \geq 0$, such that $\sum_r w_{kr} = 1$. Then, each mini-bucket r is processed using a weighted or power sum, $(\sum_{X_k} f(X)^{1/w_{kr}})^{w_{kr}}$. It is useful to note that w_{kr} can be interpreted as a "temperature"; if $w_{kr} = 1$, the power sum corresponds to a standard summation, while if $w_{kr} \to 0$, it instead corresponds to a maximization over X_k. Thus, standard mini-bucket for summation queries corresponds to choosing one mini-bucket r with $w_{kr} = 1$, and the rest with weight zero.

Weighted mini-bucket is closely related to variational bounds on the likelihood, such as conditional entropy decompositions (Globerson & Jaakkola, 2007) and tree-reweighted belief propagation (TRBP) (Wainwright, Jaakkola, & Willsky, 2005). The single-pass algorithm of Liu and Ihler (2011) mirrors standard mini-bucket, except that within each bucket a cost-shifting (or reparameterization) operator is performed, which matches the marginal beliefs (or "moments") across mini-buckets to improve the bound.

The temperature viewpoint of the weights enables a similar procedure for marginal MAP. In particular, for $X_k \in \mathbf{X}_S$, we enforce $\sum_r w_{kr} = 1$, while for $X_k \in \mathbf{X}_M$, we take $\sum_r w_{kr} = 0$ (so that $w_{kr} = 0$ for all r). The resulting algorithm, presented in Algorithm 1 and called *Weighted Mini-Bucket with Moment Matching* (WMB-MM(i)), treats MAP and sum variables differently: for sum variables it is like Liu and Ihler (2011), while taking the zero-temperature limit for MAP variables we obtain the max-marginal matching operations described for pure MAP problems in

Algorithm 1: WMB-MM(i) for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, MAP variables $\mathbf{X}_M$, constrained ordering $o = X_1, \ldots, X_n$, i-bound i
Output: Upper bound on optimal marginal MAP value

foreach $k \leftarrow n$ *downto* 1 **do**
  `// Create bucket` $\mathbf{B}_k$ `and mini-buckets` Q_{kr}
  $\mathbf{B}_k \leftarrow \{\psi_\alpha | \psi_\alpha \in \mathbf{F}, X_k \in var(\psi_\alpha)\}$; $\mathbf{F} \leftarrow \mathbf{F} \setminus \mathbf{B}_k$
  Let $\mathcal{Q} = \{Q_{k1}, \ldots, Q_{kR}\}$ be an i-partition of $\mathbf{B}_k$
  foreach $r = 1$ *to* R **do**
    $\psi_{kr} = \prod_{\psi \in Q_{kr}} \psi$; $\mathbf{Y}_r = vars(Q_{kr}) \setminus X_k$
  `// Moment Matching`
  if $X_k \in \mathbf{X}_S$ **then**
    Assign mini-bucket r weight $w_{kr} > 0$, st $\sum_r w_{kr} = 1$
    $\mu_r = \sum_{\mathbf{Y}_r} (\psi_{kr})^{1/w_{kr}}$; $\mu = \prod_r (\mu_r)^{w_{kr}}$
    Update $\psi_{kr} = \psi_{kr} \cdot \left(\frac{\mu}{\mu_r}\right)^{w_{kr}}$
  else
    $\mu_r = \max_{\mathbf{Y}_r} \psi_{kr}$; $\mu = \left(\prod_r \mu_r\right)^{1/R}$
    Update $\psi_{kr} = \psi_{kr} \cdot \left(\frac{\mu}{\mu_r}\right)$
  `// Downward Messages (eliminate` X_k`)`
  foreach $r = 1$ *to* R **do**
    if $X_k \in \mathbf{X}_S$ **then** $\lambda_{kr} \leftarrow (\sum_{X_k} (\psi_{kr})^{1/w_{kr}})^{w_{kr}}$
    else $\lambda_{kr} \leftarrow \max_{X_k} \psi_{kr}$
    $\mathbf{F} \leftarrow \mathbf{F} \cup \{\lambda_{kr}\}$
return $\prod_{\psi \in \mathbf{F}} \psi$

(Ihler et al., 2012). This mirrors the result of (Weiss, Yanover, & Meltzer, 2007), that the linear programming relaxation for MAP corresponds to a zero-temperature limit of TRBP.

The bounds provided by the single-pass WMB-MM(i) algorithm can be further tightened, if desired, by additional message passing. One example is the *Weighted Mini-Buckets with Join Graph propagation* algorithm, WMB-JG(i), which alternates between downward passes similar to Algorithm 1 and upward passes, which compute messages used to "focus" the cost-shifting in the next downward pass; see (Marinescu et al., 2014) for additional details. Another, more distributed update scheme generalizing dual decomposition methods is given in (Ping, Liu, & Ihler, 2015).

Although iterative methods can produce tighter upper bounds compared to the single-pass WMB-MM(i), when used in practice as a heuristics generator for search, the computational overhead associated with iterative schemes such as WMB-JG often outweighs the improved accuracy of its bounds (Marinescu et al., 2014). Therefore, in the sequel we use WMB-MM(i) as the heuristic to guide our various AND/OR search algorithms for marginal MAP.

4. AND/OR Search Algorithms for Marginal MAP

In this section, we introduce our search algorithms for marginal MAP. As noted, these algorithms explore the AND/OR search space for marginal MAP and are guided by effective weighted mini-bucket heuristics improved with cost shifting schemes. Specifically, we develop depth-first AND/OR branch and bound and best-first AND/OR search algorithms that explore the context minimal AND/OR search graph. Since memory can be a bottleneck, especially for best-first search, we also

develop a recursive best-first AND/OR search algorithm that can operate efficiently within bounded memory. We first review search methods for marginal MAP which precede our work.

4.1 Early Search Methods

Marginal MAP can be solved by traversing either depth-first or best-first an OR search space defined by the MAP variables. The search can be further enhanced with a specialized heuristic that upper bounds the value of each partial MAP assignment, thus allowing the use of standard pruning methods such as branch and bound. Up until recently, the best performing algorithm for marginal MAP was the depth-first branch and bound developed by Park and Darwiche (2003) which uses the unconstrained join-tree upper bound for guidance. During search, the join-tree is fully re-evaluated at each node in order to compute upper bounds for all uninstantiated MAP variables simultaneously, which allows the use of dynamic variable ordering. Although this approach provides effective bounds, the computational overhead at each node in the search space can be quite significant. Subsequently, Yuan and Hansen (2009) proposed an incremental evaluation of the join-tree bounds which significantly reduces the computational overhead. However, this modification also requires the search algorithm to follow a static variable ordering. In practice, Yuan and Hansen's method proved to be cost effective, considerably outperforming Park and Darwiche (2003). However, both methods require that the induced width of the unconstrained join tree to be small enough to be feasible.

4.1.1 DEPTH-FIRST BRANCH AND BOUND

We next describe two algorithms that generalize the Park and Darwiche (2003) and Yuan and Hansen (2009) schemes for models with high unconstrained induced width. In particular, we use MCTE(i) (Dechter et al., 2001) to approximate the exact, unconstrained join-tree inference to accommodate a maximum clique size defined by the i-bound i. The resulting branch and bound with MCTE(i) unconstrained heuristics, abbreviated BBBT[1], is given in Algorithm 2

The algorithm searches the simple OR tree of all partial MAP variable assignments. At each step, BBBT uses MCTE(i) over an unconstrained ordering to compute an upper bound $U(\bar{x})$ on the optimal extension of the current partial MAP assignment $\bar{x}$ (lines 5-8). If $U(\bar{x}) \leq L$, the current best solution cost, then the current assignment $\bar{x}$ cannot lead to a better solution and the algorithm can backtrack (line 9). Otherwise, BBBT expands the current assignment by selecting the next MAP variable in a static or dynamic variable ordering (line 4) and recursively solves a set of subproblems, one for each un-pruned domain value. Notice that when $\bar{x}$ is a complete assignment, BBBT calculates its marginal MAP value by solving a summation task over $\mathcal{M}|_{\bar{x}}$, the subproblem defined by the sum variables conditioned on $\bar{x}$ (line 2). Given sufficient resources (high enough i-bound), this can be done by variable elimination or by depth-first AND/OR search with caching (Dechter & Mateescu, 2007) (see also Section 4.3). If a better new assignment is found then the lower bound L is updated (line 10).

If MCTE(i) is fully re-evaluated at each iteration, it produces upper bounds for all uninstantiated MAP variables simultaneously. In this case, BBBT can accommodate dynamic variable orderings and can thus be viewed as a generalization of Park and Darwiche (2003). Alternatively, MCTE(i) can be done in an incremental manner to provide a generalization of Yuan and Hansen (2009); in this case BBBT requires a static variable ordering.

1. For consistency with prior work, we use the name used in Marinescu et al. (2003), (branch and bound with Bucket-Tree heuristic) to denote the same algorithm applied to pure MAP queries.

Algorithm 2: BBBT(i, $\mathbf{X}_M$, L, $\bar{x}$) for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, i-bound i, unassigned MAP variables $\mathbf{X}_M$, lower bound L, partial assignment to MAP variables $\bar{x}$
Output: Optimal marginal MAP value

```
if X_M = ∅ then
    return eval(M|_x̄)
else
    X_k ← SelectVar(X_M)
    Update MCTE(i)
    foreach value x_k ∈ D_k do
        Assign X_k to x_k:  x̄ ← x̄ ∪ {X_k = x_k}
        U(x̄) ← extract(MCTE(i))
        if U(x̄) > L then
            L = max(L, BBBT(i, X_M \ {X_k}, L, x̄))
        x̄ ← x̄ \ {X_k = x_k}
    return L
```

4.1.2 BEST-FIRST SEARCH

Best-first search schemes (e.g., A*) are known to be superior to other search schemes that traverse the same search space given the same heuristic information (Dechter & Pearl, 1985). 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 (Nilsson, 1980; Pearl, 1988). Any complete search algorithm, and in particular, depth-first branch and bound (DFBnB), must explore all the nodes expanded by A*, assuming the same tie breaking rule, and, in addition may also explore some nodes for which $f(n) < C^*$. Therefore, the expected benefit of A* over any other search algorithm, B, exploring the same search space, depends on the number of extra nodes explored by B.

To understand A*'s performance for marginal MAP 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 model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ 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 from the functions $\mathbf{F}$. Specifically, the weight of the arc $(\bar{x}_{1..l-1} \rightarrow \bar{x}_{1..l})$ is the product of all the functions $F(X_l) \subseteq F$ whose scope includes X_l and is completely instantiated by the partial assignment $\bar{x}_{1..l}$, namely $w(\bar{x}_{1..l-1} \rightarrow \bar{x}_{1..l}) = \prod_{\alpha \in F(X_l)} \psi_\alpha(\mathbf{x}_\alpha | \bar{x}_{1..l})$. The arc-weight connecting a full MAP assignment $\mathbf{x}_M$ node to s is defined by:*

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

where $F(\mathbf{X}_S) \subseteq F$ is the subset of function scopes that include summation variables in $\mathbf{X}_S$.

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.*

Proof. Let $\pi = (x_1, \ldots, x_m, s)$ be a solution path in the MMAP search space. By definition 3, the cost of π obeys $c(\pi) = \prod_{l=1}^{m} \prod_{\alpha \in F(X_l)} \psi(\mathbf{x}_\alpha | \bar{x}_{1..l}) \cdot \sum_{\mathbf{X}_S} \prod_{\alpha \in F(\mathbf{X}_S)} \psi(\mathbf{x}_\alpha | \bar{x}_{1..m})$. Let $F(\mathbf{X}_M) \subseteq F$ be the subset of function scopes that are defined on MAP variables only. It follows easily that $c(\pi) = \prod_{\alpha \in F(\mathbf{X}_M)} \psi_\alpha(\mathbf{x}_\alpha | \bar{x}_{1..m}) \cdot \sum_{\mathbf{X}_S} \prod_{\alpha \in F(\mathbf{X}_S)} \psi(\mathbf{x}_\alpha | \bar{x}_{1..m})$ and since $F = F(\mathbf{X}_M) \cup F(\mathbf{X}_S)$ we obtain $c(\pi) = \sum_{\mathbf{X}_S} \prod_{\alpha \in F} \psi(\mathbf{x}_\alpha | \bar{x}_{1..m})$. The optimal solution path π^* is $\pi^* = \operatorname{argmax}_\pi c(\pi) = \operatorname{argmax}_{\bar{x}_{1..m}} \sum_{\mathbf{X}_S} \prod_{\alpha \in F} \psi(\mathbf{x}_\alpha | \bar{x}_{1..m})$ which is the optimal solution to the 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, by a summation problem.

Thus, a search algorithm that minimizes such expansions can be significantly 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.

THEOREM **1.** *Let A* be a search algorithm exploring the OR MMAP search space and let B be any optimal search algorithm that explores the same search space as A*. Given the same heuristic function and the same tie breaking rule, we have that:*

(a) any node expanded by A will be expanded by B.*

(b) any MAP frontier node expanded by A must be expanded by B.*

Proof. When A* terminates with a path $(x_1, ..., x_m, s)$ it has found the optimal MAP solution.

Denote by R the set of full MAP assignments, of the form $(x_1, ..., x_m)$ in OPEN at the moment A* terminates (namely, those nodes which were evaluated but not expanded). Denote by S the full solutions in OPEN, having the form $(x_1, ..., x_m, s)$. Any node in S represents an evaluation of the summation sub-task. From the known optimality of A* (Nilsson, 1980), it follows that all the nodes in S must also be evaluated by B, and in particular, by DFBnB, and all have $f \geq C^*$ where C^* is the cost of an optimal MAP assignment. □

We argue that a DFBnB scheme is likely to expand *extra* MAP frontier nodes beyond S, some of which have $f < C^*$. While *extra* node expansions occur in any regular path-finding DFBnB search, they are normally easy to control. This is because DFBnB is often supplied with a near-optimal solution (e.g., using local search) which closes the gap between its best current solution L and C^*, and thus prunes the search space almost as effectively as A* (since $f(n) < L$ is close to $f(n) < C^*$). However, finding an initial good solution (and thus a good lower bound) is far harder in the context of marginal MAP as there is no simple and effective local search scheme. For these reasons, the superiority of A* over DFBnB is potentially more significant for MMAP.

4.2 AND/OR Search Spaces for Marginal MAP

The AND/OR search space for marginal MAP is defined relative to a *valid* pseudo-tree, in which the MAP variables are grouped at the top of the pseudo-tree so that they form a *start* pseudo tree, or a subtree of pseudo-tree $\mathcal{T}$ that has the same root as $\mathcal{T}$.

Given a context-minimal AND/OR graph $C_{\mathcal{T}}$ relative to a valid pseudo-tree $\mathcal{T}$, a *solution tree* $\hat{x}$ of $C_{\mathcal{T}}$ is a subtree defined over the MAP variables only such that: (1) it contains the root of $C_{\mathcal{T}}$; (2) if an internal OR node $n \in C_{\mathcal{T}}$ is in $\hat{x}$, then n is labeled by a MAP variable and exactly one of its children is in $\hat{x}$; (3) if an internal AND node $n \in C_{\mathcal{T}}$ is in $\hat{x}$ then all its OR children labeled by MAP variables are in $\hat{x}$.

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 subproblem rooted at n, while for sum variables it is the conditional likelihood (sum) of the subproblem. 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.

It is important to notice that, by definition, the AND/OR search space for marginal MAP spans both the MAP as well as the summation variables. Therefore, the AND/OR search algorithms presented in this section and the next can exploit decomposition in the MAP part of the problem as well as in the corresponding conditioned summation part.

Example 2. *Consider again the context-minimal AND/OR search graph from Figure 1(c) relative to the valid pseudo-tree from Figure 1(b). The parents sets of the variables (which define the node contexts in the search space) are shown next to the pseudo-tree nodes. It is easy to see that the MAP variables form a start pseudo-tree. A solution tree corresponding to the MAP assignment* $(A = 0, B = 1, C = 1, D = 0)$ *is indicated in red.*

4.3 AOBB: Depth-First AND/OR Branch and Bound

We are now ready to describe our depth-first AND/OR branch and bound (AOBB) algorithm for marginal MAP. The idea is to traverse the AND/OR search space in a depth-first manner and update the node values recursively based on the values of their successors. In order to guide the search more effectively, each node n in the search space that is labeled by a MAP variable is associated with a heuristic estimate $h(n)$ which provides an upper bound on $v(n)$. These estimates are then used to prune unpromising regions of the search space. AOBB is presented in Algorithm 3.

We use the notation that T'_s is the current partial solution subtree, s is the root node of the search space and the table $Cache$, indexed by node contexts, holds the partial search results. The fringe of the search is maintained by a stack called $OPEN$. The algorithm assumes that variables are selected in a static order respecting a valid pseudo-tree $\mathcal{T}$. A heuristic $f(T'_m)$ calculates an upper bound on the optimal marginal MAP extension of T'_m, where T'_m is a partial solution subtree rooted at node m (T'_m is also a subtree of T'_s and $m \in T'_s$). Each OR node n is associated with a flag $n.optimal$ which, if true, indicates that the subproblem rooted by n was solved optimally.

AOBB interleaves a top-down forward expansion of the current partial solution tree with a bottom-up cost revision step that updates the node values. It selects the tip node n on top of the search stack and, if n is labeled by a MAP variable, the algorithm attempts to prune the search space below it (lines 4–10). Specifically, it computes the heuristic evaluation function for every partial solution subtree rooted at the OR ancestors of n along the path to the root. The search below n is terminated if, for some ancestor m, $f(T'_m) \leq v(m)$, where $v(m)$ is the current lower bound on the

Algorithm 3: AOBB for marginal MAP

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

1 Create root OR node s labeled by the root X_1 of $\mathcal{T}$, set $v(s) \leftarrow -\infty$ and let $OPEN \leftarrow \{s\}$
2 **while** $OPEN \neq \emptyset$ **do**
3 Let $n \leftarrow pop(OPEN)$ and set $n.deadend \leftarrow false$
4 **if** *n is labeled by a MAP variable x_i i.e., $X_i \in \mathbf{X}_M$* **then**
5 **foreach** *OR ancestor m of n along the path to root s* **do**
6 Calculate $f(T'_m)$ using the $h(e)$ estimates of the unexpanded leaves e of T'_m
7 **if** $f(T'_m) \leq v(m)$ **then**
8 **foreach** *OR node p between n and m (excluding m)* **do**
9 Set $p.optimal \leftarrow false$
10 $n.deadend \leftarrow true$ and **break**
11 **if** $n.deadend == false$ **then**
12 **if** *n is OR node labeled by X_i* **then**
13 **if** $n.context \in Cache$ **then** $v(n) \leftarrow Cache[n.context]$
14 **else**
15 **foreach** *value $x_i \in D_i$* **do**
16 Create an AND successor n' of n labeled $\langle X_i, x_i \rangle$ and add n' to $succ(n)$
17 Initialize $h(n')$ (if $X_i \in \mathbf{X}_M$) and set $v(n') \leftarrow 1$
18 **else if** *n is AND node labeled by $\langle X_i, x_i \rangle$* **then**
19 **foreach** *children X_j of X_i in $\mathcal{T}$* **do**
20 Create an OR successor n' of n labeled $\langle X_i \rangle$, set $n'.optimal \leftarrow true$ and add n' to $succ(n)$
21 **if** $X_i \in \mathbf{X}_M$ **then** Initialize $h(n')$ and set $v(n') \leftarrow -\infty$
22 **else if** $X_i \in \mathbf{X}_S$ **then** $v(n') \leftarrow 0$
23 Add $succ(n)$ on top of $OPEN$
24 **while** $succ(n) = \emptyset$ **do**
25 Let p be the parent of n in the search space
26 **if** *n is OR node labeled X_i* **then**
27 **if** $X_i = X_1$ **then return** $v(n)$
28 **else**
29 $v(p) \leftarrow v(p) \cdot v(n)$
30 **if** $n.optimal$ **then** $Cache[n.context] \leftarrow v(n)$
31 **else if** *n is AND node labeled $\langle X_i, x_i \rangle$* **then**
32 **if** $X_i \in \mathbf{X}_M$ **then** $v(p) \leftarrow \max(v(p), v(n) + w(p, n))$
33 **else** $v(p) \leftarrow v(p) + (v(n) + w(p, n))$
34 Remove n from $succ(p)$ (and from the search space) and set $n \leftarrow p$

optimal value below m. The algorithm also labels as not optimal all of n's OR ancestors between n and m (excluding m) indicating that it is not safe to cache the values of these nodes. It is easy to see that AOBB ensures only provably optimal OR node values are cached and reused during search.

If node n was not pruned then the algorithm expands it by generating its successors. If n is an OR node, labeled by X_i, then its successors are AND nodes represented by the values x_i in variable X_i's domain (lines 12-17). Notice that at this stage the algorithm checks the cache table and if the same context $n.context$ was encountered before it is retrieved from cache which will trigger the

propagation step. If n is an AND node labeled $\langle X_i, x_i \rangle$, then its successors are OR nodes labeled by the child variables of x_i in the pseudo-tree $\mathcal{T}$ (lines 18–22).

The node values are updated by a bottom-up propagation step (lines 24–34) which is triggered when a node n has an empty set of successors. Note that as each successor is evaluated, it is removed from the set of successors in line 34. This means that all of n's children have been evaluated and their final node values are already determined. If the current node is the root, then the search terminates with the optimal marginal MAP value (line 27). If n is an OR node, then its parent p is an AND node, and therefore p updates its current value $v(p)$ by multiplication with the value of n (line 29). If node n was previously labeled optimal then the algorithm also saves in cache its value. An AND node n propagates its value to its parent p in a similar way, either by maximization or by summation depending on whether p is labeled by a MAP or by a summation variable (lines 31–33). Finally, the current node n is set to its parent p (line 34), because n was completely evaluated. Search continues either with a propagation step (if conditions are met) or with an expansion step. The algorithm can be easily instrumented to also recover the optimal solution tree $\hat{x}$ corresponding to the optimal marginal MAP value (see also Marinescu and Dechter (2009a) for additional details).

Our AOBB typically computes its heuristic $h(\cdot)$ using a weighted mini-bucket bounding scheme (see Section 3), which can be pre-compiled along the reverse order of a depth-first traversal of the valid pseudo-tree which is built along a constrained elimination order.

4.3.1 BREADTH ROTATING AOBB (BRAOBB)

Depth-first AND/OR branch and bound often lacks good anytime behavior because, during search, AOBB will solve to completion all but one independent subproblems rooted at an AND node. This behavior was first observed by Otten and Dechter (2011) in the context of pure MAP inference. Therefore, Breadth Rotating AOBB (BRAOBB) was introduced as an anytime depth-first AND/OR search scheme that rotates through different subproblems in a round-robin manner. Empirical evaluations showed that BRAOBB considerably improves AOBB's anytime behavior, and is better able to produce suboptimal solutions quickly and then gradually improve upon them. Without going into the details, the algorithm described in Otten and Dechter (2011) can be applied directly to marginal MAP queries by performing subproblem rotation to the MAP variables only.

4.4 AOBF: Best-First AND/OR Search

We next introduce a best-first search algorithm over the context-minimal AND/OR search graph (Algorithm 4). The algorithm belongs to the so-called AO* family (Nilsson, 1980). Like any AO*, AOBF maintains a partially explored AND/OR graph $C_{\mathcal{T}}$ and its current best partial solution tree T' that represents an optimal solution extension of $C_{\mathcal{T}}$ under the assumption that the tips n of $C_{\mathcal{T}}$ 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 s 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 recomputed 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. Clearly, we can use one of the best algorithms for summation subproblems such as the depth-first AND/OR search with full/adaptive

Algorithm 4: AOBF for marginal MAP

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

Insert root s in $C_{\mathcal{T}}$, where s is labeled by the root of $\mathcal{T}$
Initialize $q(s) \leftarrow h(s)$ and let $T' = \{s\}$
while *true* **do**
  Select non-terminal tip node n in the best partial tree T'.
  if $tips(T') = \emptyset$ **then break**
  `// Expand`
  foreach *successor* n' *of* n **do**
    if $n' \notin C_{\mathcal{T}}$ **then**
      Add n' as child of n in $C_{\mathcal{T}}$
    if n' *is OR node labeled by* $X \in \mathbf{X}_S$ **then**
      $q(n') \leftarrow eval(\mathcal{M}|_{T'})$
    else $q(n') \leftarrow h(n')$
  `// Update`
  foreach *ancestor* m *of* n *in* $C_{\mathcal{T}}$ **do**
    if m *is OR node* **then**
      $q(m) \leftarrow \max_{n' \in succ(m)} w(m, n') \cdot q(n')$
      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
    else $q(m) \leftarrow \prod_{n' \in succ(m)} q(n')$
  Recompute best partial tree T' by following marked arcs from the root s
return $q(s)$

caching (Mateescu & Dechter, 2007), and also allow sharing the summation sub-task computations via caching. Algorithm AOBF terminates 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 updated value of the root node.

The complexity of AOBF is bounded by the size of the underlying context minimal search graph, and is therefore time and space $O(n \cdot k^{w_c})$, where w_c is the induced width along the valid pseudo-tree (i.e., the constrained induced width) (Marinescu & Dechter, 2009a, 2009b).

4.5 RBFAOO: Recursive Best-First AND/OR Search

In practice, however, AOBF may require an enormous amount of memory, especially on difficult problem instances. We therefore developed a limited memory best-first search scheme based on the recursive best-first search idea (Korf, 1993). The algorithm is called Recursive Best-First AND/OR search with Overestimation (RBFAOO) and was shown to be very effective for pure MAP (Kishimoto & Marinescu, 2014). Algorithm 5 presents the extension of RBFAOO to the marginal MAP task.

RBFAOO uses a local threshold control mechanism to explore the context minimal AND/OR search graph in a depth first-like manner (Korf, 1993). Let the q-value $q(n)$ be an upper bound of the solution cost at node n, and let $\theta(n)$ be the threshold at n indicating the availability of a second best solution cost besides $q(n)$. RBFAOO keeps examining the subtree rooted at n until either $q(n) < \theta(n)$ or the subtree rooted at n is solved optimally. Therefore, it gradually grows the search space by updating the q-values of the internal nodes and unlike AOBF, re-expanding

Algorithm 5: RBFAOO for marginal MAP

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 s in $C_{\mathcal{T}}$, where s is labeled by the root of $\mathcal{T}$
3 `OrNode(`s, 0`)`
4 **return** $q(s)$
5 **Function** `OrNode(`n, θ`)`
6 **while** *true* **do**
7 **foreach** *AND child* c *of* n **do**
8 **if** $c.context$ *is in cache* **then** $q(c) \leftarrow ReadCache(c.context)$
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** $q(n) < \theta(n)$ *or* n *is solved* **then**
13 **break**
14 Identify two children (c_1, c_2) of n with the two highest q- values s.t. $q(c_1) \geq q(c_2) \geq q(others)$ and update threshold as $\theta(c_1) = max(\theta(n), q(c_2))/w(n, c_1)$
15 `AndNode(`c_1, $\theta(c_1)$`)`
16 $WriteCache(n.context, q(n))$
17 **Function** `AndNode(`n, θ`)`
18 **while** *true* **do**
19 **foreach** *OR child* c *of* n **do**
20 **if** $c.context$ *is in cache* **then** $q(c) \leftarrow ReadCache(c.context)$
21 **else if** c *is labeled by* $X \in \mathbf{X}_S$ **then**
22 $q(c) \leftarrow eval(\mathcal{M}|_{\bar{x}})$, where $\bar{x}$ is the assignment along the current path from root to c
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** $q(n) < \theta(n)$ *or* n *is solved* **then**
26 **break**
27 Identify an unsolved OR child c_1 of n and update threshold $\theta(c_1) = \theta(n) \cdot q(c_1)/q(n)$
28 `OrNode(`c_1, $\theta(c_1)$`)`
29 $WriteCache(n.context, q(n))$

them. The algorithm may operate with linear space (no caching). However, it can also use a fixed size cache table to store some of the nodes (based on their contexts). When RBFAOO 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). For efficiency, we implemented this step using the same recursive best-first approach but with the threshold set to $-\infty$ which essentially corresponds to depth-first AND/OR search with bounded cache.

If AOBF expands $O(N)$ nodes in the worst case, then RBFAOO expands $O(N^2)$ due to node re-expansions. Since N is bounded by $O(n \cdot k^{w_c^*})$, the time complexity of RBFAOO is bounded by $O(n^2 \cdot k^{2 \cdot w_c^*})$ but it can operate in linear space. However, in practice, by slightly overestimating the

Algorithm	Search Space	Heuristic	Variable Ordering
AOBB	AND/OR	WMB-MM(i)	Static
BRAOBB	AND/OR	WMB-MM(i)	Static
AOBF	AND/OR	WMB-MM(i)	Static
RBFAOO	AND/OR	WMB-MM(i)	Static
PARK	OR	Unconstrained join tree	Dynamic
YUAN	OR	Unconstrained join tree	Static
BBBTd	OR	MCTE(i) - unconstrained join tree	Dynamic
BBBTi	OR	MCTE(i) - unconstrained join tree	Static
OBB	OR	WMB-MM(i)	Static

Table 2: Marginal MAP search algorithms. Search spaces are either AND/OR search graph or OR search graph introduced in Section 2, and heuristic functions are defined in Section 3. Dynamic variable ordering schemes re-evaluate the heuristic function as search proceeds, whereas static schemes only evaluate it once before search.

node thresholds (by a small constant δ), RBFAOO avoids such a high node re-expansion overhead (see also Kishimoto and Marinescu (2014) for more details).

4.6 Experimental Results

We next evaluate empirically the performance of the proposed depth-first and best-first AND/OR search algorithms as exact schemes for solving optimally the marginal MAP task. All competing algorithms were implemented in C++, and the experiments were conducted on a cluster with 27 computing nodes. Each node has a dual-core 2.67 GHz Intel Xeon X5650 CPU with 24 GB of RAM and was operated by a 64-bit Linux operating system. Each run of algorithms was allotted a 2-hour time limit and a 24 GB memory limit.

4.6.1 OVERVIEW AND METHODOLOGY

Algorithm Setup Table 2 shows all algorithms evaluated in the following experiments. We considered the AND/OR algorithms AOBB, AOBF, BRAOBB, and RBFAOO, respectively. They all use the weighted mini-bucket heuristic WMB-MM(i) (see Section 3) which has a single parameter called i-bound that controls the accuracy at the cost of using memory exponential in i. The hyperparameters of BRAOBB and RBFAOO are given as follows: we set the rotation limit to 1,000 for BRAOBB and the overestimation parameter to 1.0 for RBFAOO, respectively. The best-first AND/OR algorithms, AOBF and RBFAOO, use 4 GB of memory for storing the values of nodes in the AND/OR search graph. The depth-first AND/OR algorithms, AOBB and BRAOBB, use cache tables up to the total memory limit. We compare the AND/OR search algorithms with the OR search algorithms, PARK/YUAN and their generalizations BBBTd/BBBTi, respectively. We also evaluate the depth-first OR search algorithm with the WMB-MM(i) heuristic that we call OBB in order to compare the impact of using the AND/OR search space. For this purpose, we modified AOBB to explore an OR search space by enforcing a chain structure on the MAP variables in the guiding pseudo-tree.

Benchmarks The benchmark problem sets are derived from the Pascal2 competition problems (Elidan et al., 2012). They include (1) the *grid* domain which is a collection of random grid networks

Benchmark		# inst	n	k	w_c	h_c	w_u	h_u
grid	easy	15	144 – 1156	2 – 2	16 – 52	50 – 164	15 – 49	48 –198
	hard	60	144 – 1156	2 – 2	25 – 375	42 – 421	–	–
pedigree	easy	10	334 – 1289	4 – 7	35 – 237	51 – 134	15 – 29	60 –160
	hard	40	334 – 1289	4 – 7	35 – 237	63 – 259	–	–
promedas	easy	10	453 – 1849	2 – 2	10 – 122	42 – 174	10 – 106	43 –157
	hard	40	453 – 1849	2 – 2	11 – 490	36 – 507	–	–

Table 3: Benchmark instances. $\#$. inst is the number of instances in each domain. We also distinguish easy and hard instances. The minimum and the maximum values from the set of problems are shown in the following parameters: n is the number of variables, k is the maximum domain size, w_c is the constrained induced width, h_c is the height of the pseudo-tree corresponding to the constrained elimination ordering. The unconstrained induced width, w_u and pseudo-tree height, h_u are also shown to highlight the difficulty of hard marginal MAP problem instances.

ranging from 12-by-12 to 34-by-34 grids, (2) the *pedigree* domain which is a collection of genetic linkage analysis problems that are related to the haplotyping task, and (3) the *promedas* domain which is a collection of Markov networks modeling an expert system for medical diagnosis tasks. Since the original problems from the Pascal2 competition define pure MAP tasks, we generated synthetic marginal MAP problem instances from each MAP problem as follows. For each problem instance, we selected m variables and marked them as MAP variables while the remaining ones were marked as summation variables, respectively. Specifically, we generated two types of marginal MAP instances with m MAP variables for each network, easy and hard. The easy instances were generated by selecting the first m variables from a breadth-first traversal of a pseudo-tree obtained from a join tree decomposition of the primal graph and the hard instances were generated by selecting the MAP variables uniformly at random. The easy instances were designed so that problem decomposition is maximized and the constrained and unconstrained elimination orders are relatively close to each other, thus having similar induced widths. In contrast, the hard instances tend to have very large constrained induced widths w_c compared to the easy problem instances and to the unconstrained induced width w_u. In our experiments, m is selected to be equal to 50% of the variables (see also Table 3 for more details). For consistency, all competing algorithms used the same constrained or unconstrained elimination ordering.

Measures of Performance From the evaluation results, we report the following metrics: (1) the CPU time in seconds for finding an optimal marginal MAP value from selected instances, (2) the number of summation problems evaluated until finding an optimal solution from selected instances, (3) the number of instances that returned an optimal solution from benchmark domains, and (4) the average CPU time for finding an optimal solution from benchmark domains. The first metric compares the CPU time for search algorithms that terminate with an optimal solution on an individual instance with varying strength of the heuristic functions controlled by the i-bound. The second metric compares the number of summation subproblems evaluated until finding an optimal solution on an individual instance with varying i-bounds, especially for the AND/OR search algorithms since

the exact computation of summation subproblem imposes significant overheads to AND/OR search algorithms. The third metric visualizes the fraction of problem instances solved optimally given the total time and memory resources, and the last metric complements the third metric by comparing the average CPU time spent on finding an optimal solution from the common subset of benchmark domains that are solved by all AND/OR search based algorithms.

4.6.2 THE CPU TIME FOR FINDING THE OPTIMAL SOLUTION

Figure 2 shows the CPU time in seconds for finding the optimal solution on selected problem instances from the grid, pedigree and promedas domains, respectively. In each plot, we show only the algorithms that were able to prove optimality within the 2-hour time limit. Clearly, we see that the AND/OR search algorithms dominate all the OR search algorithms. The curves of AOBB, AOBF, BRAOBB, and RBFAOO lie below the curves of PARK, YUAN, BBBTd, BBBTi, and OBB at all i-bounds on all 6 instances. Note, the absence of a curve means that the algorithm could not terminate within the 2-hour time limit.

- PARK/YUAN failed to find an optimal solution for all problem instances except the *easy pedigree1* instance due to high unconstrained induced widths. The unconstrained induced width w_u of each problem instance is $w_u = 29$ for *grid 75-21-5*, $w_u = 17$ for *pedigree1*, and $w_u = 27$ for *or_chain_4fg*, respectively. Therefore, PARK/YUAN couldn't generate the unconstrained join-tree based heuristic for the *grid 75-21-5* and *or_chain_4fg* instances because of the memory limit.
- BBBTd/BBBTi solved additional instances compared to PARK/YUAN, e.g., the *easy or_chain_4fg* was solved in 216 seconds and in 142 seconds by BBBTd and BBBTi, respectively. This is due to the use of the MCTE(i) heuristic that partitions the unconstrained join-tree with the i-bound.
- AOBB, AOBF, BRAOBB, and RBFAOO found an optimal solution in about 10 seconds with i-bound 16 or higher on these problem instances. Furthermore, on the *hard or_chain_4fg* instance, BBBTd and BBBTi found the optimal solution in 2863 seconds and 6832 seconds, respectively, while AOBB, AOBF, BRAOBB, and RBFAOO only spent 626 seconds, 293 seconds, 744 seconds, and 117 seconds, respectively, at the same i-bound 18.
- RBFAOO found an optimal solution on the *easy grid 75-21-5*, the *hard grid 75-21-5*, the *hard pedigree1*, and the *hard or_chain_4fg* instances with the smallest reported i-bounds (from 10 to 14) when other AND/OR search algorithms failed to find an optimal solution due to the time or memory limit.

We can see that algorithms AOBB and BRAOBB consistently dominate the OBB counterpart on all instances and at all reported i-bounds. Since OBB and AOBB are based on the same algorithm with the only difference being in how the search space is generated, we can conclude that the performance gain of AOBB and BRAOBB compared to OBB can be attributed to the compactness of AND/OR search space. When we compare OBB with other OR search algorithms we can see that OBB is the fastest OR search algorithm for finding an optimal solution. For example, OBB finds the optimal solution faster than BBBTd/BBBTi at the i-bound 18 or higher on the *easy or_chain_4fg* and *hard or_chain_4fg* instances. Furthermore, OBB is the only OR search algorithm that terminated within the 2 hour time limit on the *easy grid 75-21-5* and the *hard pedigree1* instances. Since OR search algorithms are only distinguished by the heuristic functions, we can also conclude that for

(a) `grid`

instance (n, f, k, w_c, w_u)	algorithm	$i=10$ time	$i=10$ nodes	$i=12$ time	$i=12$ nodes	$i=16$ time	$i=16$ nodes	$i=18$ time	$i=18$ nodes
grid easy	PARK	167				1458			
75-16-5 I0	YUAN	29				17119			
(256, 256, 2, 22, 21)	BBBTd	oot	302716	659	54520	42	17491	50	19584
	BBBTi	oot	176156	2345	70590	85	30898	39	31424
	OBB	1401	100538171	3	215404	0.48	7779	0.49	1733
	AOBB	65	1919431	0.52	22334	0.41	3253	0.43	646
	BRAOBB	7	566677	0.59	27771	0.35	2832	0.44	758
	AOBF	7	212165	0.38	10687	0.27	2672	0.28	583
	RBFAOO	**1**	290917	**0.22**	22279	**0.16**	2042	**0.26**	693
grid hard	PARK	1415				5810			
75-16-5 I1	YUAN	508				483065			
(256, 256, 2, 82, 21)	BBBTd	oot	360097	oot	40291	4546	2389	2114	1607
	BBBTi	oot	1009103	oot	115935	6288	3139	oot	3220
	OBB	oot	220938105	oot	243066878	oot	319929239	85	4351922
	AOBB	oot	398833504	632	37708909	**5**	340611	3	224719
	BRAOBB	6964	380238949	324	18873588	**5**	350248	3	227999
	AOBF	**728**	15144899	721	13471576	10	249319	7	196812
	RBFAOO	7161	553985425	**309**	29811222	7	784408	**2**	229745

(b) `pedigree`

instance (n, f, k, w_c, w_u)	algorithm	$i=10$ time	$i=10$ nodes	$i=12$ time	$i=12$ nodes	$i=16$ time	$i=16$ nodes	$i=18$ time	$i=18$ nodes
pedigree easy	PARK	oom				oom			
pedigree33 I0	YUAN	oom				oom			
(798, 798, 4, 28, 24)	BBBTd	oot	208815	oot	145663	oot	52744	oot	43948
	BBBTi	oot	246582	oot	209620	oot	120843	oot	50220
	OBB	oot	72686306	3584	31265034	726	5896706	745	527116
	AOBB	20	877731	7	328531	4	107914	5	25357
	BRAOBB	33	1498544	11	532770	5	157469	7	38222
	AOBF	14	473329	7	212385	3	63495	4	17158
	RBFAOO	**10**	1082112	**4**	450426	**2**	107853	**4**	29505
pedigree hard	PARK	oom				oom			
pedigree 33 I3	YUAN	oom				oom			
(798, 798, 4, 90, 24)	BBBTd	oot	129028	oot	50439	oot	9883	oot	7514
	BBBTi	oot	17291423	oot	11827308	oot	2208894	oot	15783
	OBB	oot	90882192	oot	88027605	oot	92162950	oot	108961101
	AOBB	oot	371082807	oot	398810432	3064	194413004	334	20981934
	BRAOBB	oot	252364162	oot	274187876	oot	274849031	700	32731707
	AOBF	oom	17448194	oom	16341293	oom	15891613	oom	11957878
	RBFAOO	oot	653477484	**3751**	333264049	**880**	86995051	**55**	6515774

(c) `promedas`

instance (n, f, k, w_c, w_u)	algorithm	$i=10$ time	$i=10$ nodes	$i=12$ time	$i=12$ nodes	$i=16$ time	$i=16$ nodes	$i=18$ time	$i=18$ nodes
promedas easy	PARK	oom				oom			
or chain8 fg I0	YUAN	oom				oom			
(1044, 1055, 2, 63, 56)	BBBTd	oot	123551	oot	28609	3191	17816	oot	13992
	BBBTi	oot	1202749	oot	85250	oot	25481	oot	13975
	OBB	oot	45854883	oot	44042781	oot	24204379	oot	21909874
	AOBB	oot	430376779	oot	359556370	oot	391666636	1583	108529398
	BRAOBB	oot	359396003	oot	312111365	5288	272736813	758	57647876
	AOBF	oom	16843900	oom	17038664	oom	17191924	510	15923902
	RBFAOO	**6425**	555895245	**1268**	126337093	**682**	78371234	**152**	21786146
promedas hard	PARK	oom				oom			
or chain4 fg I4	YUAN	oom				oom			
(691, 701, 2, 100, 27)	BBBTd	3496	4178	607	2650	261	3009	591	2031
	BBBTi	4319	31507	4798	12221	2463	3699	1545	3295
	OBB	oot	54940954	oot	60336495	oot	80233378	oot	84429447
	AOBB	oot	388478093	oot	403472921	497	30889027	55	4205131
	BRAOBB	3320	199196858	2127	147963142	154	12173651	33	2754174
	AOBF	630	17599487	550	17612588	100	3276123	44	1505211
	RBFAOO	**215**	29608915	**326**	44855073	**26**	4260266	**9**	1443645

Table 4: CPU time in seconds and number of nodes expanded for solving selected problem instances. Time limit 2 hours, memory limit 24GB. 'oot' and 'oom' stand for out-of-time and out-of-memory, respectively.

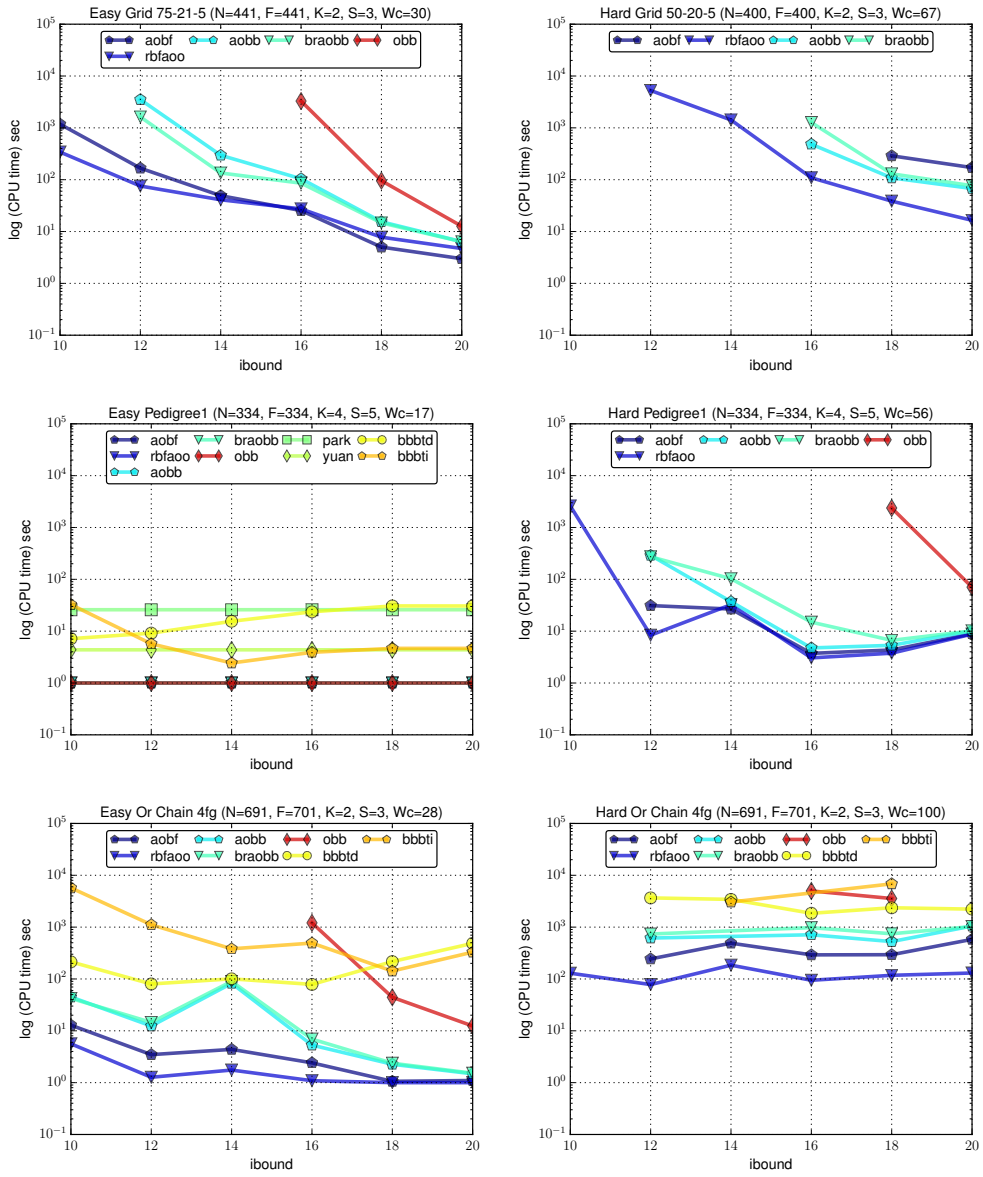


Figure 2: The CPU time in seconds from 6 problem instances. The horizontal axis is the i-bound of WMB-MM(i) heuristic for AND/OR search algorithms and MCTE(i) heuristic for OR search algorithms. The vertical axis is the CPU time in log scale. The missing data points indicate a search algorithm failed to find the optimal solution in time limit of 2 hours. PARK/YUAN are shown only in the easy pedigree1 instance since both failed to find the optimal solution for all other instances in the time limit, and BBBTi/BBBTd are also shown in the easy pedigree1 instance and promedas instances.

the same i-bound value the WMB-MM(i) heuristic is tighter than the MCTE(i) heuristic used by BBBTd/BBBTi.

When comparing the AND/OR search algorithms, we can conclude that the best-first search algorithms perform better than the depth-first search ones on all problem instances and at all i-bounds according to the current metric. Specifically, RBFAOO the variant that operates within limited memory, emerges as the best performing exact AND/OR search algorithm. Table 4 reports additional results on individual problem instances from the 3 benchmark domains. The results display similar patterns as before, namely the AND/OR search algorithms dominate the OR search algorithms.

4.6.3 THE NUMBER OF CONDITIONAL LIKELIHOOD EVALUATIONS

Figure 3 shows the number of conditional likelihood evaluations by the AND/OR search algorithms from 6 problem instances at varying i-bounds. Note that we only show the number of conditional likelihood evaluations when a search algorithm found the optimal solution. Thus, the missing data points in the plot indicates a search algorithm couldn't find the optimal solution within the 2-hour time limit. For example:

- On the *easy grid 75-21-5* instance with i-bound 18, AOBF, RBFAOO, AOBB and BRAOBB evaluated 282, 173, 3106, and 1924 summation subproblems, respectively. The trend is similar on the *hard grid 75-21-5* instance as well.
- On the *hard grid 50-20-5* instance with i-bound 18, AOBF, RBFAOO, AOBB and BRAOBB evaluated 226474, 55355, 481760, and 572050 summation subproblems.
- On the *hard pedigree 9* instance with i-bond 18, AOBF, RBFAOO, AOBB and BRAOBB evaluated 2985, 4838, 6410, and 21632 summation subproblems.

We can see that AOBF and RBFAOO consistently evaluate a smaller number of summation problems than AOBB and BRAOBB. It is important to note that RBFAOO has a much smaller computational overhead compared with AOBF which often translates into a much higher node expansion rate per second than that of AOBF. This means that RBFAOO is able to reach the most promising region of the search space (containing the optimal solution) quicker than AOBF which explains the reduced number of conditional likelihood evaluations of RBFAOO. We discuss next the two major sources of AOBF's significant additional overhead. First, when expanding the current node n, AOBF may need to update the node values of all n's ancestors all the way up to the root, whereas RBFAOO may only update the q-value of n's parent. Second, AOBF needs to maintain in memory the entire explicated search graph and this may be difficult especially when the memory used approaches the 24GB limit. In contrast, RBFAOO conducts the search in a depth-first like manner and uses only 4GB of RAM for its cache table. We also observe that higher i-bounds lead to fewer summation evaluations for all search algorithms because of improved heuristics and more effective pruning.

4.6.4 THE NUMBER OF INSTANCES THAT RETURNED AN OPTIMAL SOLUTION

Figure 4 provides aggregated results showing the number of instances on which the algorithms returned an optimal solution for all 3 problem domains combining easy and hard problem instances. We observe that the AND/OR search algorithms solved a larger number of instances than the OR search algorithms on all domains. Moreover, the best-first AND/OR search algorithms, AOBF and RBFAOO, solved more instances than the depth-first AND/OR search algorithms, AOBB and BRAOBB, respectively.

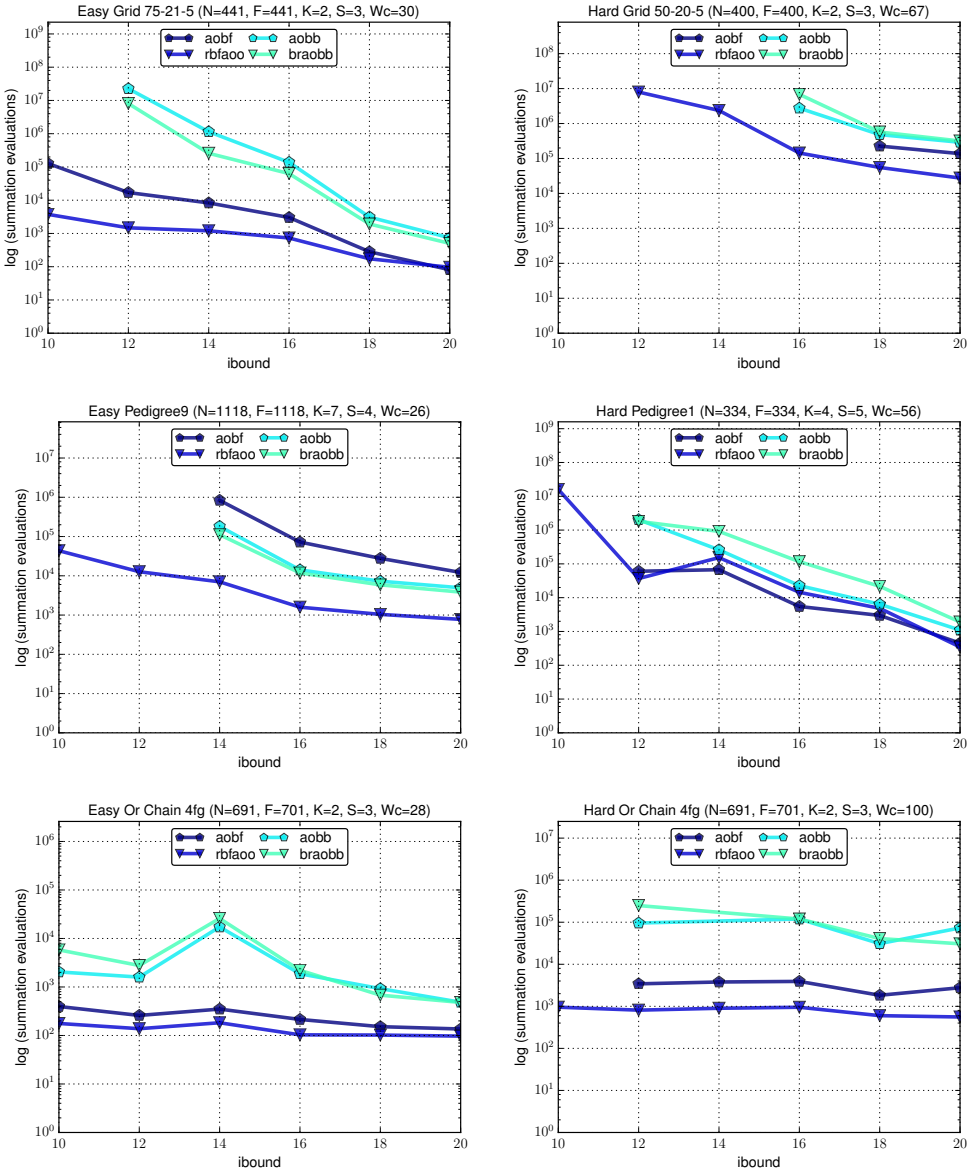


Figure 3: The number of conditional likelihood evaluations of AND/OR search algorithms from 6 problem instances. The horizontal axis is the i-bound of WMB-MM(i) heuristic for AND/OR search algorithms and MCTE(i) heuristic for OR search algorithms. The vertical axis is the number of summation problems solved until finding the optimal solution within 2 hour time limit.

- On the `grid` domain with i-bound 20, AOBF, RBFAOO, AOBB, BRAOBB, OBB, BBBTd, and BBBTi solved 76%, 79%, 73%, 76%, 55%, 28%, and 23% of the problem instances, respectively.

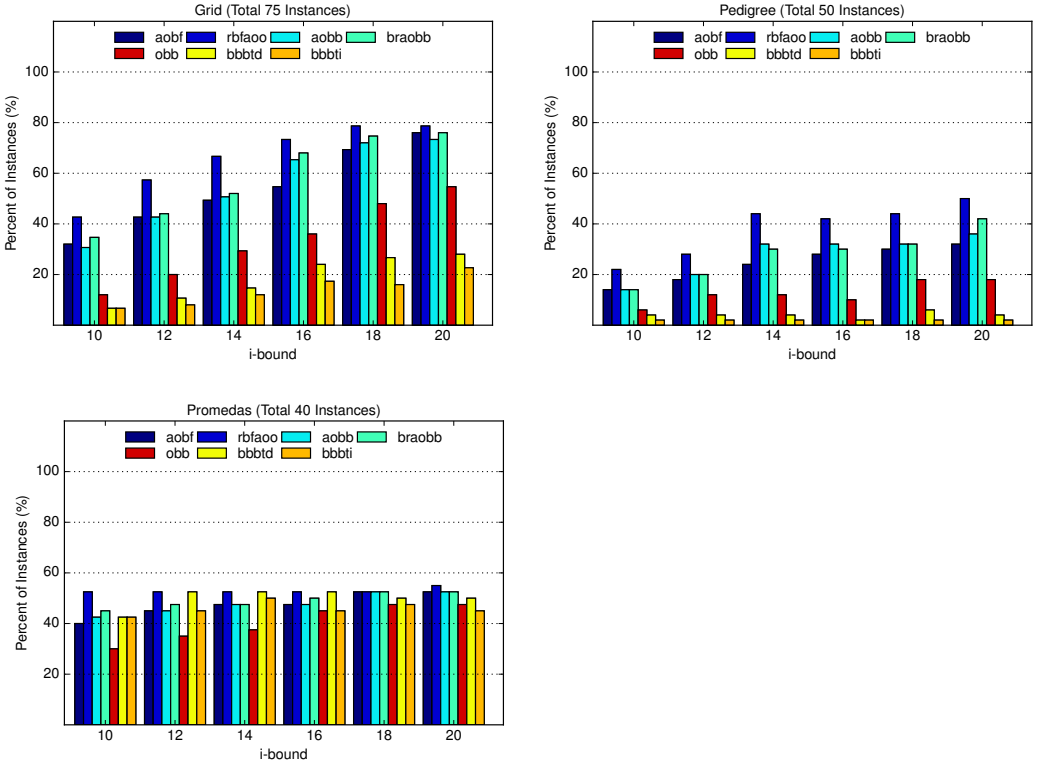


Figure 4: The number of instances returned with the optimal solution from 3 problem domains. Easy and hard problem instances are combined into a single group. The horizontal axis is the i-bound of WMB-MM(i) heuristic function and the vertical axis is the normalized count on the instances returned with the optimal solution in 2 hour time limit. AND/OR search algorithms solved more instances than OR search algorithms, and RBFAOO solved the largest number of instances at all domains over all i-bounds.

- On `pedigrees` with i-bound 20, AOBF, RBFAOO, AOBB, BRAOBB, OBB, BBBTd, and BBBTi solved 32%, 50%, 36%, 42%, 18%, 4%, and 2% following the similar trend as in the `grid` domain.
- On `promedas` with i-bound 20, AOBF, RBFAOO, AOBB, BRAOBB, OBB, BBBTd, and BBBTi solved 52%, 55%, 53%, 53%, 47%, 50%, and 40%, showing that RBFAOO still performs the best but the difference between AND/OR search algorithms and OR search algorithms are less significant than on the other two domains.

Clearly, RBFAOO is the best performing algorithm on all domains and at all reported i-bounds. Comparing AOBF and AOBB, both algorithms solved a similar number of instances in the 2-hour time limit and the 24 GB memory limit, while BRAOBB solved slightly fewer instances than AOBB because of the computational overhead associated with rotating over the independent subproblems in the AND/OR search space. BBBTd and BBBTi solved the smallest number of problem instances across all domains and both failed to solve even a single instance from the `hard pedigree`

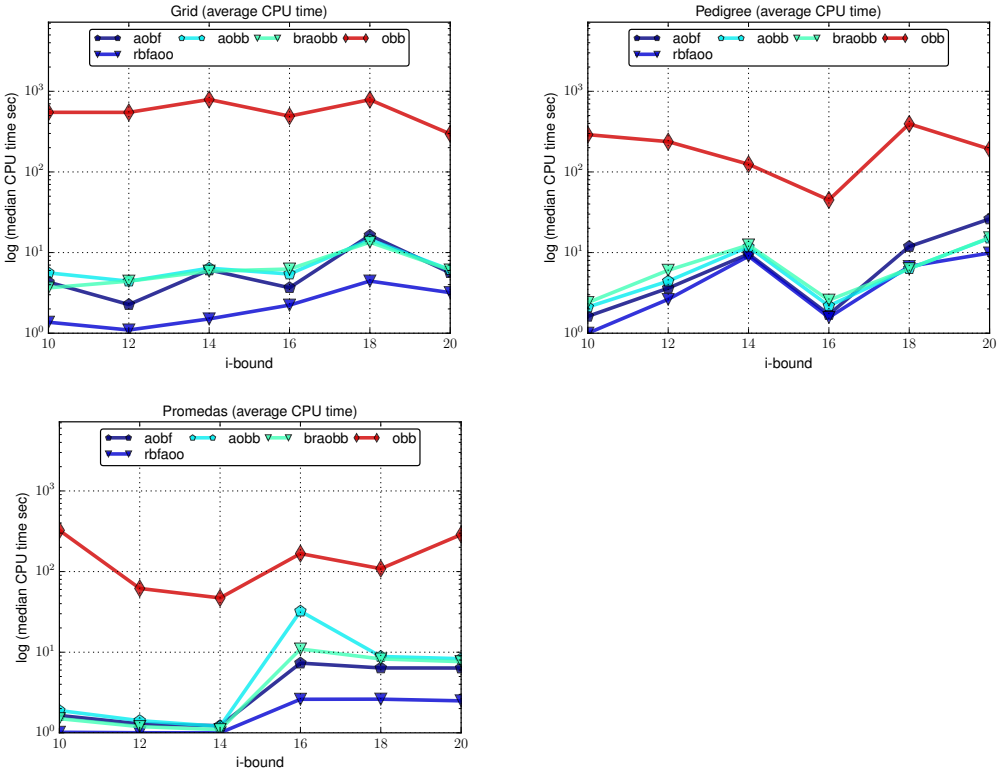


Figure 5: The average CPU time for finding the optimal solutions in a benchmark. The horizontal axis is the i-bound of WMB-MM(i) heuristic function and the vertical axis is the average CPU time for finding the optimal solution in log scale. The average time was computed from the common problem instances that are solved by all four algorithms in two hours. The best-first search spent less time than depth-first search. RBFAOO is the fastest algorithm across all problem domains and i-bounds.

instances. The performance of OBB lies between AND/OR search algorithms and the other OR search algorithms guided by unconstrained join-tree based heuristics, thus demonstrating on one hand the effectiveness of the WMB-MM(i) heuristics compared to the MCTE(i) with unconstrained order, and the compactness of the AND/OR search space versus the OR search space on the other hand.

4.6.5 THE AVERAGE CPU TIME

Since the previous metric only considers the number of solvable instances (thus providing the provable optimal solution), we report in Figure 5 the average CPU time in seconds for finding the optimal solution, for problem instances solved by all four AND/OR search algorithms to avoid bias: 40 grid instances, 9 pedigree instances, and 19 promedas instances were solved optimally by all four algorithms with i-bound 20.

- On the `grid` domain with i-bound 20, the average CPU times of AOBF, RBFAOO, AOBB, BRAOBB, and OBB search algorithms are 5.61 seconds, 3.18 seconds, 6.18 seconds, 6.07 seconds, and 296.11 seconds, respectively. RBFAOO and AOBF solved problem instances faster than BRAOBB and AOBB, and OBB spent significantly longer time compared to the AND/OR search algorithms.

- On `pedigrees` with i-bound 20, the average CPU times of AOBF, RBFAOO, AOBB, BRAOBB, and OBB search algorithms are 26.13 seconds, 9.87 seconds, 15.09 seconds, 15.26 seconds, and 192.05 seconds, respectively. We see again a similar trend as before.

- On `promedas` with i-bound 20, the average CPU times of AOBF, RBFAOO, AOBB, BRAOBB, and OBB search algorithms are 6.36 seconds, 2.49 seconds, 8.30 seconds, 7.63 seconds, and 287.27 seconds, respectively.

In summary, for exact algorithms, we conclude that RBFAOO is overall the best exact search algorithm. We also see that AOBF can solve problem instances faster than AOBB and BRAOBB at higher i-bounds, while AOBB and BRAOBB are competitive with AOBF at smaller i-bounds.

5. Anytime Search Algorithms for Marginal MAP

In the previous section we introduced the main *exact* search schemes traversing the AND/OR space for marginal MAP. These schemes have obvious shortcomings. Best-first search provides a solution only at termination. Depth-first search is more flexible with anytime capabilities and seems more tuned with our goal of yielding anytime performance as well as bounds that can be tightened with time. In this section we show how we can move towards both these goals by building upon the complementary properties of best-first and depth-first search. We first explore the potential of the well-known principle of *weighted search* that converts best-first search algorithms into anytime search schemes (Section 5.1). This scheme was previously explored for pure MAP (Flerova et al., 2017), and we will show in the sequel how it can be extended to marginal MAP. Subsequently, (Section 5.2), we will present an alternative anytime approach that combines the best-first and depth-first principles into hybrid search schemes. Both methods provide anytime confidence intervals in the form of improving upper and lower bounds on the optimum.

5.1 Weighted Best-First AND/OR Search

Weighted best-first AND/OR search was introduced recently as an effective alternative to anytime depth-first search schemes for pure MAP tasks (Flerova et al., 2017). These algorithms were evaluated on a wide range of benchmark problems and were shown to be highly competitive with Breadth Rotating AOBB search (Otten & Dechter, 2011), one of the best performing anytime MAP solvers.

We first note that in this section (and only in this one) we will convert the marginal MAP task into a minimization task, because weighted heuristic schemes are normally, and more naturally described in the context of min-sum tasks. We show next how to transform Equation 1 into a minimization problem. Let F_M be the subset of functions whose scopes are subsumed by MAP variables only, i.e., $F_M = \{\alpha \,:\, \mathbf{X}_\alpha \subseteq \mathbf{X}_M\}$. Then, we have that:

$$
\begin{aligned}
\mathbf{x}^* &= \underset{x_M}{\operatorname{argmin}} -\log\left(\sum_{\mathbf{X}_s}\prod_{\alpha\in F}\psi_\alpha(\mathbf{x}_\alpha|x_M)\right)\\
&= \underset{x_M}{\operatorname{argmin}} -\log\left(\sum_{\mathbf{X}_s}\prod_{\alpha\in F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\cdot\prod_{\alpha\in F\setminus F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\right)\\
&= \underset{x_M}{\operatorname{argmin}} -\log\left(\prod_{\alpha\in F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\cdot\sum_{\mathbf{X}_s}\prod_{\alpha\in F\setminus F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\right)\\
&= \underset{x_M}{\operatorname{argmin}} -\log\left(\prod_{\alpha\in F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\right) - \log\left(\sum_{\mathbf{X}_s}\prod_{\alpha\in F\setminus F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\right)\\
&= \underset{x_M}{\operatorname{argmin}} \sum_{\alpha\in F_M} -\log\psi_\alpha(\mathbf{x}_\alpha|x_M) - \log\left(\sum_{\mathbf{X}_s}\prod_{\alpha\in F\setminus F_M}\psi_\alpha(\mathbf{x}_\alpha|x_M)\right)\\
&= \underset{x_M}{\operatorname{argmin}} \sum_{\alpha\in F_M}\theta_\alpha(\mathbf{x}_\alpha|x_M) + \phi(x_M)
\end{aligned}
\tag{2}
$$

It is easy to see that $\phi(x_M)$ corresponds to the summation subproblem conditioned on the assignment x_M to the MAP variables $\mathbf{X}_M$. Therefore, algorithms AOBF and RBFAOO can be modified in a straightforward manner to solve Equation 2 by replacing maximization with minimization at the OR nodes labeled by MAP variables, and multiplication with summation at their AND children, respectively. We also note that for each of the conditioned summation subproblems the algorithms consider the negative log of the respective value.

Yet, note that when the arc-costs are larger than 1, the negative log-transformation yields negative numbers (e.g. Markov networks), which are not suitable for the min-sum formulation. It may be possible to handle this by normalizing by a large constant. Since our benchmarks are all based on Bayesian networks we did not encounter this problem and did not explore nor experimented with this issue and we leave this for future work.

5.1.1 WEIGHTED AOBF AND RBFAOO

The fixed-weighted version of the AOBF and RBFAOO algorithms can be obtained by multiplying the heuristic function $h(n)$ of a node n in the AND/OR search graph by a weight $\alpha > 1$ (i.e., substituting $h(n)$ by $\alpha \cdot h(n)$). This would involve only the portion of the search space that corresponds to the MAP variables. If $h(n)$ is admissible, which is the case for mini-bucket heuristics, then the cost of the solution obtained by weighted AOBF (or weighted RBFAOO) is α-optimal, namely it is guaranteed to be within a factor α from the optimal one. These schemes extend well known approaches such as WA* (Pohl, 1970) and WAO* (Chakrabarti, Ghose, & Sarkar, 1987) to the marginal MAP query.

5.1.2 ITERATIVE WEIGHTED AOBF WITH REPAIRS

Since weighted AOBF yields α-optimal solutions, it can then be extended into an iterative anytime scheme that we call WAOBF (Algorithm 6), by decreasing the weight from one iteration to the next (until it becomes 1). Specifically, WAOBF starts by running weighted AOBF with an initial weight

Algorithm 6: WAOBF for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$, $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$, heuristic $h(\cdot)$, initial weight α_0
Output: α**-optimal** marginal MAP value $\mathcal{V}(\bar{x})$, and assignment $\bar{x}$

1 Initialize $\alpha = \alpha_0$, $\mathcal{V}(\bar{x}) = -\infty$, $\bar{x} = \emptyset$
2 **while** $\alpha >= 1$ **do**
3 $\quad (\mathcal{V}(\bar{x}'), \bar{x}') \leftarrow$ **AOBF**$(\mathcal{M}, \alpha \cdot h)$
4 $\quad$ Maintain and report current best solution as $(\mathcal{V}(\bar{x}), \bar{x})$
5 $\quad$ Decrease weight α according to schedule policy
6 **return** $(\mathcal{V}(\bar{x}), \bar{x})$

Algorithm 7: WAOBF-REP for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$, $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$, heuristic $h(\cdot)$, initial weight α_0
Output: α**-optimal** marginal MAP value $\mathcal{V}(\bar{x})$, and assignment $\bar{x}$

1 Create root OR node s labeled by root of $\mathcal{T}$
2 Initialize $\alpha = \alpha_0$, $C'_{\mathcal{T}} = \{s\}$, $\mathcal{V}(\bar{x}) = -\infty$, $\bar{x} = \emptyset$, $T = \emptyset$
3 **while** $\alpha >= 1$ **do**
4 $\quad (\mathcal{V}(\bar{x}'), \bar{x}', C'_{\mathcal{T}}) \leftarrow$ **AOBF**$(\mathcal{M}, \alpha \cdot h, C'_{\mathcal{T}}, T)$
5 $\quad$ Maintain and report current best solution as $(\mathcal{V}(\bar{x}), \bar{x})$
6 $\quad$ Decrease weight α according to schedule policy
7 $\quad$ Revise $C'_{\mathcal{T}}$ according to $\alpha \cdot h$ from leaves to the root
8 $\quad$ Update the best partial solution tree T from revised $C'_{\mathcal{T}}$
9 **return** $(\mathcal{V}(\bar{x}), \bar{x})$

α_0 until it finds a suboptimal solution. It then restarts the search with a decreased weight using a weight update schedule policy such as $\alpha_{i+1} = \sqrt{\alpha_i}$. Clearly, different schedules can be used, however, the latter proved quite effective in practice when solving a MAP query (Flerova et al., 2017). Notice that the algorithm also outputs a α-optimality guarantee for the smallest α it reached.

Running WAOBF afresh at each iteration seems redundant however. Therefore, we introduce another anytime scheme, called WAOBF-REP, that is based on the anytime repairing AOBF approach for pure MAP (Flerova et al., 2017). The algorithm is capable of reusing information from previous iterations. WAOBF-REP is described in Algorithm 7. As noted, it does not discard the explicated AND/OR search graph $C'_{\mathcal{T}}$ after finishing the i-th iteration but rather revises the node values in $C'_{\mathcal{T}}$ from leaves to the root node according to the newly inflated heuristic $\alpha_{i+1} \cdot h(n)$. Then, search restarts with the revised best partial solution tree.

5.1.3 ITERATIVE WEIGHTED RBFAOO

Finally, an anytime weighted RBFAOO, called WRBFAOO, is obtained by replacing the call to AOBF in line 3 of Algorithm 6 with a call to RBFAOO, respectively.

5.2 Hybrid Best+Depth-First AND/OR Search

While weighted best-first search schemes turned out to be favorable for marginal MAP compared with depth-first search, as we will show in our experiments (see also Lee et al. (2016)), their anytime performance seems quite sensitive to the initial weight and to the weight update schedule. In particular: (i) the algorithms sometimes spend significant computational resources (time, memory) struggling to find even the first suboptimal solution; (ii) depending on the weight update schedule,

either very few solutions are generated or there is no improvement in the solution quality; (iii) most critically, the sub-optimality bound produced by the weight is often very loose.

We now explore schemes that combine best-first and depth-first search into best+depth hybrid anytime search algorithms. These algorithms provide *anytime upper and lower bounds* on the optimal marginal MAP value that can be used to better gauge the solution quality during search, and these bounds tighten with time.

5.2.1 NOTATIONS

Let $C'_{\mathcal{T}}$ denote the explicated context-minimal AND/OR search graph relative to pseudo-tree $\mathcal{T}$. Note that in this case $C'_{\mathcal{T}}$ is defined over the MAP variables only. Each node $n \in C'_{\mathcal{T}}$ maintains two values $q(n)$ and $l(n)$, respectively. The quantity $q(n)$ represents an upper bound provided by the heuristic evaluation function at the node n (i.e., the weighted mini-bucket value), while $l(n)$ is the cost of the current best solution found below node n, and is therefore a lower bound on the best solution in the search space below n. We use $\mathcal{U}$ and $\mathcal{L}$ to denote the current best global upper and lower bounds on the optimal MMAP of the root node. For node $n \in C'_{\mathcal{T}}$, $ch(n)$ denote its children in $C'_{\mathcal{T}}$, while $w_{(n,m)}$ is the weight labeling the arc $n \rightarrow m$ in $C'_{\mathcal{T}}$. Algorithm 8 describes the node expansion (`EXPAND(`n`)`) and node values update (`UPDATE(`n`)`) procedures during search. Clearly, the q- and l-values are updated bottom-up based on the corresponding values of their children in the search graph. Note that during the update of q-values, we also mark with a $\star$ symbol the arc corresponding to the best child m' of an OR node n.

5.2.2 LAOBF: BEST-FIRST AND/OR SEARCH WITH DEPTH-FIRST LOOKAHEADS

As noted before, the rationale behind best+depth based search is to alternate in some manner between the two styles of search space exploration, where the best-first component progresses towards improved upper bounds and depth-first progresses towards improved potential solutions and their accompanying lower bounds.

A simple way to augment best-first search with a mechanism that generates solutions is to do explicit depth-first lookahead dives under some nodes in the best-first search frontier (Stern, Kulberis, Felner, & Holte, 2010). Our first best+depth hybrid, *Best-First AND/OR Search with Depth-First Lookaheads* (LAOBF), is described in Algorithm 9. We elaborate on these dives next.

DFS and BFS Iterations Let T_b be the current best partial solution tree of the best-first search scheme, $tips(T_b)$ be the set of tip nodes of T_b, and m be one of these tips. The algorithm performs a depth-first dive at the subtree rooted at m, recording the conditioned lower bound found as $l(m)$. This is accomplished by the `dfs-lookahead(`m`)` function in line 8. Once all the tips of T_b are explored in such a depth-first manner, a global lower bound $\mathcal{L}$ of T_b can be obtained by multiplying the generated lower bounds $l(m)$ by their corresponding arc costs in T_b (line 10). Once such a full dive is accomplished, best-first search takes over as usual, expanding a tip node n from the current T_b, updating the q-values of n's ancestors, and selecting the next best partial solution tree. Then, a new depth-first dive can be performed from all tip nodes, and so on. The updated q-value of the root node provides an anytime (and often improved) upper bound $\mathcal{U}$ on the optimal MMAP value.

Performing the depth-first lookaheads at every single iteration can often incur significant overhead. To bound this overhead, we use a *cutoff* parameter θ to trigger the depth-first lookaheads every θ best-first iterations (see line 6). It is also important to note that the cache information is shared

Algorithm 8: Expanding a node and updating the node q- and l-values (LAOBF and AAOBF)

Input: node n, pseudo-tree $\mathcal{T}$, search graph $C'_{\mathcal{T}}$, heuristic $h(\cdot)$

1 **Function** EXPAND (n) **:**
2 **if** *n is OR node labeled by $\langle X_i \rangle$ and $X_i \in \mathbf{X}_M$* **then**
3 **foreach** *values $x_i \in D_i$* **do**
4 Create AND child c labeled by $\langle X_i, x_i \rangle$ and add c to $C'_{\mathcal{T}}$ if not already there
5 **if** *c is terminal* **then**
6 $q(c) \leftarrow 1$ and $l(c) \leftarrow 1$
7 **else**
8 $q(c) \leftarrow h(c)$ and $l(c) \leftarrow -\infty$
9 **else if** *n is AND labeled by $\langle X_i, x_i \rangle$ and $X_i \in \mathbf{X}_M$* **then**
10 **foreach** *successors X_j of X_i in $\mathcal{T}$* **do**
11 Create OR child c labeled by $\langle X_j \rangle$ and add c to $C'_{\mathcal{T}}$ if not already there
12 **if** $X_j \in \mathbf{X}_M$ **then**
13 $q(c) \leftarrow h(c)$ and $l(c) \leftarrow -\infty$
14 **else if** $X_j \in \mathbf{X}_S$ **then**
15 $q(c) = l(c) \leftarrow eval(\mathcal{M}|\bar{\mathbf{x}})$
16 **Function** UPDATE (n) **:**
17 **forall** *ancestor p of n in $C'_{\mathcal{T}}$, including n* **do**
18 **if** *p is OR node* **then**
19 $l(p) \leftarrow \max_{m \in ch(p)}(w_{(p,m)} \cdot l(m))$
20 $q(p) \leftarrow \max_{m \in ch(p)}(w_{(p,m)} \cdot q(m))$
21 $m' \leftarrow \operatorname{argmax}_{m \in ch(p)}(w_{(p,m)} \cdot q(p))$
22 Mark with symbol $\star$ the arc $p \rightarrow m'$
23 **else if** *p is AND node* **then**
24 $l(p) \leftarrow \prod_{m \in ch(p)} l(m)$
25 $q(p) \leftarrow \prod_{m \in ch(p)} q(m)$

between the depth-first and best-first iterations, so that the solutions to the summation subproblems can be reused.

5.2.3 AAOBF: ALTERNATING BEST-FIRST WITH DEPTH-FIRST AND/OR SEARCH

LAOBF described above restricts the depth-first exploration to the subspaces below the tip nodes of the current best partial solution tree T_b, and, as we will show in the experimental section, this may not always lead to improved lower bounds. Moreover, we find that LAOBF requires carefully tuning the cutoff parameter, which may be challenging. Therefore, our second approach called *Alternating Depth-First and Best-First AND/OR Search* (AAOBF) is a parameter-free best+depth hybrid that aims to diversify the depth-first exploration (thus allowing it to explore feasible solutions in a greedier manner, rather than restricting it to specific regions dictated by the best-first search process).

Specifically, AAOBF (Algorithm 10) alternates between a depth-first and a best-first stage and maintains two independent partial solution trees for each. In its depth-first stage it expands, depth-first, a current *feasible* partial solution tree T_l (lines 4–12), where T_l is a partial assignment that can be extended to a feasible (often suboptimal) solution, to improve the global lower bound. In its best-first stage it expands, best-first, a current *best* partial solution tree T_b (lines 13–30) to improve

Algorithm 9: LAOBF for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$, heuristic function $h(\cdot)$, $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$, cutoff θ
Output: Anytime upper and lower bounds on optimal marginal MAP value

1 Create an OR node s labeled by the root of $\mathcal{T}$
2 Initialize $\mathcal{U} = q(s) = h(s)$, $\mathcal{L} = l(s) = -\infty$, $C'_{\mathcal{T}} = \{s\}$, $T_b = \{s\}$
3 $counter \leftarrow 0$
4 **while** $\mathcal{U} \neq \mathcal{L}$ **do**
5 Let $cost(T_b) = \prod_{(n,m) \in arcs(T_b)} w_{(n,m)}$ i.e., the product of arc costs in T_b
6 **if** $counter$ ***mod*** $\theta = 0$ **then**
7 **forall** *node* m *in* $tips(T_b)$ **do**
8 $l(m) \leftarrow$ `dfs-lookahead(`m`)`
9 **if** $cost(T_b) \cdot \prod_{m \in tips(T_b)} l(m)) > \mathcal{L}$ **then**
10 $\mathcal{L} \leftarrow cost(T_b) \cdot \prod_{m \in tips(T_b)} l(m))$
11 $\mathcal{U} \leftarrow q(s)$
12 **print**$\langle \mathcal{U}, \mathcal{L} \rangle$
13 Select non-terminal tip node n in T_b
14 EXPAND(n)
15 UPDATE(n)
16 Select new T_b by tracing down $\star$-marked arcs from root s
17 $counter \leftarrow counter + 1$
18 **return** $\mathcal{U}$

the global solution and an associated upper bound and guide the exploration closer to the region containing the optimal solution. At any stage during search, the partial solution trees T_l and T_b can be identified by tracing down $\diamond$- and $\star$-marked arcs (respectively) from the root s of $G'_{\mathcal{T}}$.

The algorithm begins by expanding non-terminal tip nodes from the current T_l. Each node expansion is followed by a bottom-up revision of the q- and l-values (lines 6–7). An OR node which was just expanded also marks with a $\diamond$ symbol the arc to its best AND child (lines 8–10). Notice that the $\diamond$-markings do not change during this stage, and therefore T_l is extended depth-first to a solution tree.

When a solution is found (i.e., T_l has no tips), AAOBF attempts to give control to best-first search. Before turning to best-first search, it revises the $\diamond$-markings along the arcs in $G'_{\mathcal{T}}$ so that a new feasible partial solution tree could be selected later. Specifically, if there is a lower bound $l(n)$ at node n (i.e., $l(n) \neq -\infty$) then AAOBF marks with a $\diamond$ symbol the arc to the AND child that minimizes the ratio between the current $l(n)$ and the successor's updated q-value, and therefore is the most likely to increase $l(n)$ (line 19). Otherwise, it selects the best AND child having the largest heuristic (upper bounding) value (line 21).

AAOBF continues by expanding nodes from T_b in the usual best-first manner. However, if during this stage a new feasible partial solution tree T_l is identified (line 29), the algorithm switches immediately to the depth-first expansion of the new T_l. Search terminates with the optimal marginal MAP value when the global lower and upper bounds are equal.

Since the proposed best+depth-first search algorithms for marginal MAP traverse the context-minimal AND/OR search graph, we have that:

THEOREM **2** (complexity)**.** *Algorithms LAOBF and AAOBF are sound and complete (namely, they will find an optimal solution if given enough time and space). Their complexity is time* $O(n \cdot m \cdot k^{w_c^*}$

Algorithm 10: AAOBF for marginal MAP

Input: Graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$, pseudo-tree $\mathcal{T}$, heuristic function $h(\cdot)$, $\mathbf{X}_M = \mathbf{X} \setminus \mathbf{X}_S$
Output: Anytime upper and lower bounds on optimal marginal MAP value

Create an OR node s labeled by the root of $\mathcal{T}$
Initialize $\mathcal{U} = q(s) = h(s)$, $\mathcal{L} = l(s) = -\infty$, $C'_{\mathcal{T}} = \{s\}$, $T_l = \{s\}$, $T_b = \{s\}$, $flag = false$
while $\mathcal{U} \neq \mathcal{L}$ **do**
  if $tips(T_l) \neq \emptyset$ **then**
    Select non-terminal tip node n in T_l
    EXPAND(n)
    UPDATE(n)
    if n *is OR node* **then**
      $m' \leftarrow \text{argmax}_{m \in ch(n)}(w_{(n,m)} \cdot q(m))$
      Mark with symbol $\diamond$ the arc $n \rightarrow m'$
    Select new T_l using $\diamond$-marked arcs from s
    $flag \leftarrow true$
  else
    if $l(s) > \mathcal{L}$ **then**
      $\mathcal{U} \leftarrow q(s)$, $\mathcal{L} \leftarrow f(s)$ and **print** $\langle \mathcal{U}, \mathcal{L} \rangle$
    if $flag = true$ **then**
      forall *OR nodes* n *in* $C'_{\mathcal{T}}$ **do**
        if $l(n) \neq -\infty$ **then**
          $m' \leftarrow \text{argmin}_{m \in ch(n)} \frac{l(n)}{w_{(n,m)} \cdot q(m)}$
        else
          $m' \leftarrow \text{argmax}_{m \in ch(n)}(w_{(n,m)} \cdot q(m))$
        Mark with symbol $\diamond$ the arc $n \rightarrow m'$
      Select new T_b using $\star$-marked arcs from s
      $flag \leftarrow false$
    Select non-terminal tip node n in T_b
    EXPAND(n)
    UPDATE(n)
    Select new T_b using $\star$-marked arcs from s
    Select new T_l using $\diamond$-marked arcs from s
return $\mathcal{U}$

and space $O(n \cdot k^{w_c^*})$, where n is the total number of variables, m is the number of MAP variables k bounds the domain size, and w_c^* it the induced width of the valid pseudo-tree (i.e., constrained induced width).

Proof. Clearly, the size of the search space explored, and therefore the space complexity of both LAOBF and AAOBF, is bounded by $O(n \cdot k^{w_c^*})$. For both LAOBF and AAOBF, each node expanded during the best-first search stage can be followed in the worst case by a greedy depth-first dive to compute a complete (but not optimal) MAP assignment. Since the depth-first dive takes $O(m)$ time, we have that the time complexity of algorithms LAOBF and AAOBF is $O(n \cdot m \cdot k^{w_c^*})$. □

Finally, we note that both LAOBF and AAOBF can switch to using only depth-first tree search as soon as memory fills up, thus continuing to improve the lower bound.

5.3 Experimental Results

In this section, we report results obtained with the anytime search algorithms on the same benchmark domains and using the same evaluation environment as in Section 4.6.

5.3.1 OVERVIEW AND METHODOLOGY

We evaluate algorithms WAOBF, WAOBF-REP, WRBFAOO, LAOBF and AAOBF using the weighted mini-bucket based heuristic function WMB-MM(i). The hyperparameters of the algorithms are set as follows. The initial weight α_0 used by the weighted best-first AND/OR search algorithms WAOBF, WAOBF-REP and WRBFAOO is 64, while the weight update schedule is $\alpha_{i+1} = \sqrt{\alpha_i}$. The cutoff parameter of LAOBF is 1000.

Since the anytime AND/OR search algorithms generate anytime suboptimal solutions (lower bounds) as well as anytime upper bounds, we developed the anytime performance metrics that focus on the improvement of the quality of anytime solutions and upper bounds over time. In the following subsections, we first show plots depicting the anytime solutions and upper bounds for selected problem instances. Then, we report 3 aggregate metrics to compare the anytime performance of the anytime AND/OR search algorithms, as follows: (1) the number of instances that returned any solution at various time bounds, which compares how fast an algorithm can generate any solution, (2) the average of relative anytime solution qualities as a function of time, which compares the quality of anytime solutions, and (3) the average gap as a function of time, which compares the gap between the upper bound and the anytime solution found by each algorithm. The first metric characterizes how fast an algorithm can generate any solution, regardless of the quality. This fast response time is a desirable property in many real-time applications. The second metric complements the first one by considering the relative solution quality over time, while the last metric characterizes the average quality of anytime solutions by deviations from the corresponding upper bounds.

5.3.2 RESULTS ON INDIVIDUAL INSTANCES

We now present the anytime solutions and upper bounds found on selected problem instances from each benchmark. For the weighted best-first AND/OR search algorithms, we compute upper bounds using the weight. Recall that these algorithms guarantee that at termination the generated solution lies within a constant factor α of the optimal solution. In Figure 6, we show plots of the anytime solutions and upper bounds as a function of time for i-bounds 12 and 18, respectively. Table 5 reports similar numerical results obtained on additional problem instances.

- On the `grid` domain, the anytime weighted AND/OR search algorithms perform significantly worse than the hybrid AND/OR search algorithms. For example, in Table 5 we see that on the *hard grid 90-34-5* instance, all three weighted AND/OR search algorithms failed to find a single suboptimal solution due to the memory limit, while AAOBF and LAOBF found anytime solutions and upper bounds at the 1 minute time bound and improved them over time. Furthermore, on the *hard grid 90-30-5* instance, only WAOBF returned any solution within the 1 minute time bound, but AAOBF and LAOBF found better anytime solutions and upper bounds, respectively. We also observe that both AAOBF and LAOBF generate frequent anytime updates compared to the weighted best-first search algorithms.
- On `pedigrees`, both algorithms AAOBF and LAOBF found and improved the anytime solutions and upper bounds over time on the *hard pedigree38*, *hard pedigree39*, and *hard*

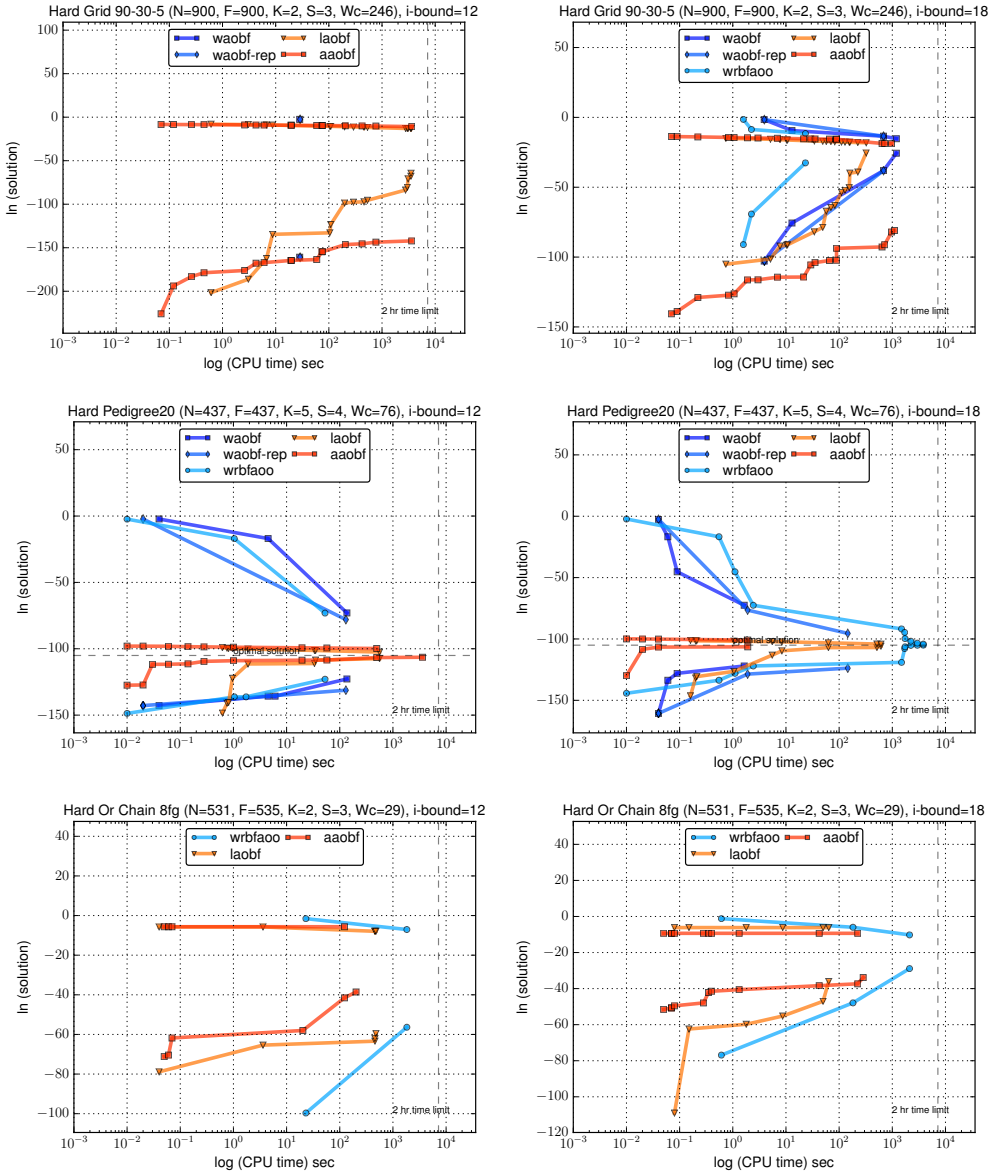


Figure 6: Anytime lower and upper bounds from hard problem instances with i-bound 12 (left) and 18 (right). The horizontal axis is the CPU time in log scale and the vertical axis is the value of marginal MAP in log scale. This figure visualizes the improvement of anytime solutions over increased time bounds up to 2 hour, and each data point of the lower curves corresponds to the time that anytime solutions were discovered. The upper curves show the upper bounds. A search algorithm terminates with the optimal solution when upper and lower curves coincided.

pedigree20 instances. We see WAOBF-REP failed to find a single solution on the *hard pedigree38* and the *hard pedigree39* instances. WAOBF also failed on the *hard pedigree38*

instance due to the memory limit. We can also observe that AAOBF and LAOBF converge faster to the optimal solution on the *hard pedigree20* instance.

- On promedas, algorithms WAOBF, WAOBF-REP, and WRBFAOO failed to find a single solution on the *hard or_chain16_fg* and the *hard or_chain24_fg* instances, while both AAOBF and LAOBF found and improved the solutions and upper bounds over time. Moreover, AAOBF and LAOBF produced better anytime solutions than BRAOBB on all reported problem instances with i-bounds 12 and 18, respectively.

Overall, from Figure 6 and Table 5, we see that the anytime solutions (lower curves) found by AAOBF and LAOBF are superior to those found by WAOBF, WAOBF-REP, and WRBFAOO, respectively. Furthermore, the distinction of the quality of anytime solution is clear at i-bound 12. Comparing AAOBF and LAOBF, we can see that AAOBF found higher quality solutions at earlier time bounds on the *hard pedigree20* and the *hard or_chain8_fg* instances. In addition, the hybrid AND/OR search algorithms produced more frequent anytime solutions compared to the weighted best-first AND/OR search algorithms. WAOBF found only a single anytime solution on the *hard grid 90-30-5* instance at i-bound 12, while AAOBF and LAOBF found 14 and 15 anytime solution within the same 2-hour time limit, respectively. We also observe that LAOBF and AAOBF produced tighter upper bounds than the weighted best-first search algorithms. Comparing the upper bounds produced by LAOBF and AAOBF, we see that they mostly overlap on the problem instances considered.

5.3.3 THE NUMBER OF INSTANCES THAT RETURNED ANY SOLUTION AT TIME T

We next report the number of instances that returned any solution at various time bounds. This metric visualizes how fast each algorithm can produce a solution, regardless of its quality. Figure 7 compares the percentage of problem instances returning feasible solutions at varying time bounds for all competing algorithms at the i-bounds 12 and 18, respectively. Note that the dashed lines and the solid lines correspond to the ratio computed from anytime solutions and optimal solutions, respectively.

In Figure 7, we see that RBFAOO solves optimally within time the largest number of instances on all benchmark and at all reported i-bounds. However, at the 1 minute time bound and for i-bound 18, the percent of instances exactly solved by RBFAOO is only 61%, 42%, and 36% on the grid, pedigree, and promedas domain, respectively. Considering the fact that the exact best-first AND/OR search algorithms are usually terminated by the memory limit, the anytime AND/OR search algorithms are more desirable for improving the coverage with suboptimal solutions. Comparing the dashed lines, we observe that both AAOBF and LAOBF cover more instances than the weighted best-first AND/OR search algorithms at all reported time bounds.

- On the grid domain with i-bound 18, AAOBF and LAOBF covered 98.6% and 97.3% of the instances at the 1 minute time bound, while WAOBF, WAOBF-REP, and WRBFAOO covered 93.3%, 93.3%, and 92.0%, respectively. BRAOBB also generated anytime solutions for 97.3% of the instances. We can see that all anytime AND/OR search algorithms solved more than 30% of the instances compared to RBFAOO.
- On pedigrees with i-bound 18, AAOBF and LAOBF covered 92.0% and 80.0% of the instances, respectively, at the 1 minute time bound, while WAOBF, WAOBF-REP, and WRBFAOO covered 80.0%, 80.0%, and 84.0%, respectively. BRAOBB covered 54.0% of the

(a) grid

instance (n, f, k, w_c, w_u)	algorithm	$i = 12$				$i = 18$			
		1min		1hr		1min		1hr	
		lb	ub	lb	ub	lb	ub	lb	ub
grid hard	AAOBF	-163.581	-9.54662	-143.601	-10.8083	-103.973	-15.563	-80.962	**-18.8718**
90-30-5 I1	LAOBF	-134.606	**-10.0975**	**-64.4327**	**-13.1226**	**-67.161**	**-17.2402**	**-25.4589**	-18.5986
(900, 900, 2, 246, 43)	WAOBF	-160.909	-2.5142	-	-	-75.6271	-9.45339	-25.663	-15.2593
	WAOBF-REP	-	-	-	-	-	-	-	-
	WRBFAOO	-	-	-	-	-	-	-	-
	BRAOBB	**-126.653**	-	-126.653	-	-104.404	-	-104.404	-
grid hard	AAOBF	-235.169	1.23546	-224.245	0.937635	-96.3262	-11.9388	-71.4547	-12.7446
90-34-5 I1	LAOBF	**-194.728**	**-0.52338**	**-170.317**	**-0.85827**	**-80.75**	**-14.599**	**-54.0588**	**-15.346**
(1156, 1156, 2, 301, 49)	WAOBF	-	-	-	-	-	-	-	-
	WAOBF-REP	-	-	-	-	-	-	-	-
	WRBFAOO	-	-	-	-	-	-	-	-
	BRAOBB	-253.812	-	-253.812	-	-118.629	-	-106.33	-

(b) pedigree

instance (n, f, k, w_c, w_u)	algorithm	$i = 12$				$i = 18$			
		1min		1hr		1min		1hr	
		lb	ub	lb	ub	lb	ub	lb	ub
pedigree hard	AAOBF	-216.404	-161.012	-214.364	-161.309	**-183.628**	-163.778	**-181.372**	-164.04
pedigree38 I1	LAOBF	**-208.81**	**-162.96**	**-198.402**	**-163.595**	-199.539	**-164.345**	-198.698	**-164.942**
(724, 724, 5, 149, 62)	WAOBF	-	-	-	-	-	-	-	-
	WAOBF-REP	-	-	-	-	-	-	-	-
	WRBFAOO	-	-	-285.035	-35.6294	-	-	-184.065	-109.446
	BRAOBB	-232.987	-	-212.611	-	-206.441	-	-206.441	-
pedigree hard	AAOBF	-321.692	-299.275	-320.595	-300.202	-325.879	-301.936	-315.208	-302.987
pedigree39 I1	LAOBF	**-321.044**	**-301.09**	**-319.8**	**-303.019**	**-318.817**	**-304.423**	**-313.72**	**-306.28**
(1272, 1272, 5, 132, 20)	WAOBF	-334.015	-118.092	-332.306	-197.591	-329.458	-116.481	-329.458	-116.481
	WAOBF-REP	-	-	-	-	-	-	-	-
	WRBFAOO	-343.759	-204.401	-343.759	-204.401	-340.981	-202.749	-332.914	-256.712
	BRAOBB	-	-	-	-	-	-	-	-

(c) promedas

instance (n, f, k, w_c, w_u)	algorithm	$i = 12$				$i = 18$			
		1min		1hr		1min		1hr	
		lb	ub	lb	ub	lb	ub	lb	ub
promedas hard	AAOBF	-226.479	**-18.7189**	-210.327	**-18.7195**	-137.205	**-19.853**	-136.494	**-19.8553**
or chain 16 fg I1	LAOBF	**-192.147**	-18.5613	**-191.184**	-18.5617	**-129.625**	-9.39193	**-120.084**	-9.39332
(1675, 1701, 2, 490, 90)	WAOBF	-	-	-	-	-	-	-	-
	WAOBF-REP	-	-	-	-	-	-	-	-
	WRBFAOO	-	-	-	-	-	-	-	-
	BRAOBB	-218.756	-	-218.756	-	-190.695	-	-190.695	-
promedas hard	AAOBF	**-111.666**	-7.1599	**-88.933**	-7.16056	**-64.8236**	**-10.0497**	**-64.8236**	**-10.0497**
or chain 24 fg I1	LAOBF	-129.625	**-9.39193**	-120.084	**-9.39332**	-125.953	-9.79024	-95.841	-9.81755
(1155, 1172, 2, 154, 71)	WAOBF	-	-	-	-	-	-	-	-
	WAOBF-REP	-	-	-	-	-	-	-	-
	WRBFAOO	-	-	-	-	-	-	-	-
	BRAOBB	-166.674	-	-115.428	-	-113.823	-	-113.823	-

Table 5: Lower and upper bounds on the optimal value on 2 selected instances from each benchmark at 1 minute and 1 hour time bound with i-bound 12 and 18. In the table, *lb* denotes the solution cost at the time bound and *ub* denotes the upper bound. The highlighted results show best performance for an instance and a time bound. The hybrid schemes seem superior.

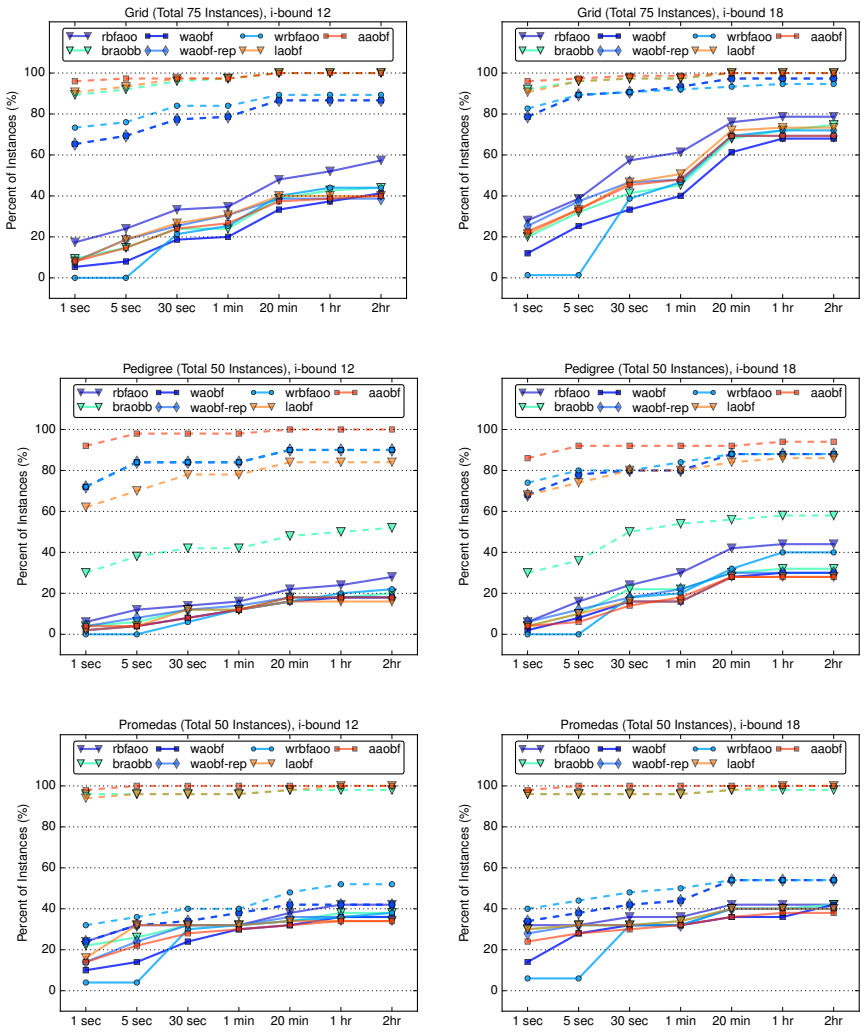


Figure 7: The number of instances returned any solution at time t with the i-bound 12 (left) and 18 (right). The horizontal axis is time bounds from 1 second to 2 hours and the vertical axis is the percent of instances solved. The solid lines show the progress of finding optimal solutions with increased time bounds by anytime AND/OR search algorithms. The dashed line corresponds to percent of instances that returned at least one solution within the time bounds. Anytime AND/OR search shows good performance in is ability for finding feasible solutions in a shorter time bounds. In particular, AAOBF and LAOBF returned feasible solutions for more than 90 % of instances on all domains in 5 second time bounds. Overall, AAOBF and LAOBF outperformed other algorithms on this metric.

instances. Compared to RBFAOO, the anytime AND/OR search algorithms covered more than 50% of the instances.

- On promedas with i-bound 18, the coverage of AAOBF and LAOBF is 100.0% and 96.0% at the 1 minute time bound, while WAOBF, WAOBF-REP, and WRBFAOO covered 44.0%, 44.0%, and 54.0%, respectively. BRAOBB covered 90.0 % of the instances. The trend is quite different in the promedas domain where AAOBF and LAOBF showed more than 60% improvement over RBFAOO while WAOBF, WAOBF-REP, and WRBFAOO improved over RBFAOO only by 10%.

Overall, we can conclude that AAOBF and LAOBF are the best performing algorithms according to the current metric.

5.3.4 THE AVERAGE OF RELATIVE ANYTIME SOLUTION QUALITIES AT TIME T

We next report our second aggregate measure; the average of relative solution qualities as a function of time. We define the relative solution quality of an anytime search algorithm on a measure per instance i at time t by:

$$\rho^i_{sol}(t) = \frac{\text{The solution cost of algorithm at time } t}{\text{The best solution cost found at time } t \text{ by any algorithm}} \tag{3}$$

Larger values indicate superior solutions (i.e., larger lower bounds). When all anytime search algorithms failed to produce a single anytime solution, we set $\rho^i_{sol}(t)$ to 0. The average of relative anytime metric is the mean of $\rho^i_{sol}(t)$ over all problem instances a benchmark domain:

$$\rho_{sol}(t) = \frac{\sum_{i \in I} \rho^i_{sol}(t)}{|I|}, \tag{4}$$

where I is the set of instances and $|I|$ denotes the total number of instances in a domain.

Figure 8 reports the average relative solution quality obtained on the grid, pedigree and promedas domains, using i-bounds 12 and 18, respectively. In this case, RBFAOO can only take two values for $\rho_{sol}(t)$, namely either 1.0 if it terminated within the time bound or 0.0 otherwise. We observe that all algorithms show superior performance compared with RBFAOO on all benchmark and at almost all reported time bounds.

- On the grid domain with i-bound 18 at the 1 minute time bound, the solution qualities of AAOBF, LAOBF, WAOBF, WAOBF-REP, WRBFAOO, and BRAOBB are 0.935, 0.926, 0.887, 0.894, 0.923, and 0.836, respectively. We can see that both AAOBF and LAOBF maintain a higher solution quality compared to the weighted AND/OR search algorithms. In this case BRAOBB performs the worst. The trend is similar for the smaller i-bound of 12.
- On pedigrees with i-bound 18 at the 1 minute time bound, the solution qualities of AAOBF, LAOBF, WAOBF, WAOBF-REP, WRBFAOO, and BRAOBB are 0.931, 0.799, 0.802, 0.794, 0.836, and 0.528, respectively. Here, AAOBF is still superior while LAOBF and the weighted best-first search algorithms perform similarly to each other. BRAOBB is inferior to AAOBF and the gap remains consistent over time. Again, the trend is similar for i-bound 12.
- On promedas domain with i-bound 18 at the 1 minute time bound, the solution qualities of AAOBF, LAOBF, WAOBF, WAOBF-REP, WRBFAOO, and BRAOBB are 0.907, 0.841, 0.414, 0.415, 0.460, and 0.877, respectively. We can see that AAOBF and LAOBF maintain higher solution quality compared to the weighted AND/OR search algorithms. The improvement of

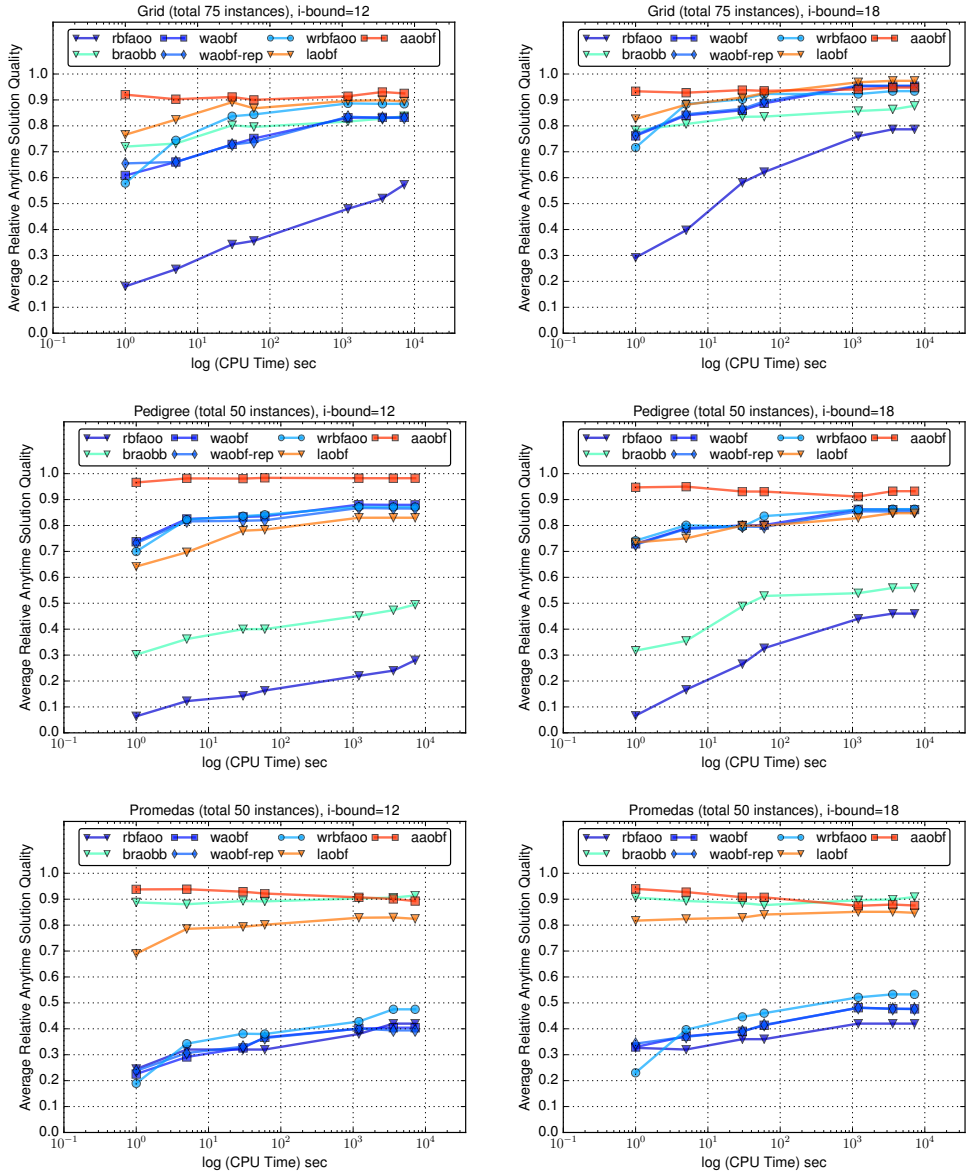


Figure 8: The average relative anytime solution qualities with i-bound 12 (left) and 18 (right). The horizontal axis is the time bounds from 1 second to 2 hour and the vertical axis is the relative anytime solution quality ranging from 0 to 1. Each line shows the changes of the solution quality over varying time bounds for a search algorithm. On the average, the hybrid search algorithms obtain the best solution quality in most cases and the quality measure approaches 1.0.

the solution quality achieved by weighted best-first search compared to RBFAOO is not as significant (< 0.1). We also see that BRAOBB is the best performing algorithm at longer time

bounds. For example, the solution quality at the 2-hour time bound of BRAOBB and AAOBF is 0.909 and 0.876, respectively.

Overall, we see that AAOBF and LAOBF produced higher quality anytime solutions than the other anytime AND/OR search algorithms across benchmarks and i-bounds. The weighted best-first search algorithms have similar relative solution quality but WRBFAOO consistently outperforms WAOBF and WAOBF-REP, respectively.

5.3.5 THE AVERAGE GAP OF ANYTIME SOLUTIONS AND UPPER BOUNDS AT TIME T

The last metric considered is the average gap for a benchmark domain at time t. We define the gap at time t for algorithm A as the ratio between the logarithm of the upper bound $\mathcal{U}$ generated by A and the logarithm of the solution cost $\mathcal{L}$ (lower bound) at time t, as follows:

$$\gamma_A(t) = \begin{cases} \frac{\alpha-\beta}{\alpha} & \text{if } 0 \leq \mathcal{L},\ \mathcal{U} \leq 1 \\ \frac{\alpha}{\alpha-\beta} & \text{if } 0 \leq \mathcal{L} \leq 1 \text{ and } 1 \leq \mathcal{U} \end{cases} \tag{5}$$

where $\alpha = \log(\text{solution cost at } t)$ and $\beta = \log(\text{upper bound at } t)$. For the weighted best-first search algorithms, we set $\beta = \frac{\alpha}{w}$ and always use the first formula in Equation 5. Therefore, $0 \leq \gamma_A(t) \leq 1$, and $\gamma_A(t)$ is 0 only if an anytime algorithm founds an optimal solution. When an algorithm failed to find a single anytime solution from an individual instance, we set $\gamma_A(t)$ to 1.0. The average gap is the mean of $\gamma_A(t)$ over all problem instances in the benchmark domain.

Figure 9 shows the average gap between the upper and the lower bounds of the algorithms that are able to provide both upper and lower bounds, using i-bounds 12 and 18, respectively. Therefore, algorithms RBFAOO and BRAOBB are excluded.

- On the `grid` domain with i-bound 18 at the 1 minute time bound, the average gap qualities of AAOBF, LAOBF, WAOBF, WAOBF-REP, and WRBFAOO are 0.127, 0.095, 0.144, 0.124, and 0.126, respectively. We observe that LAOBF is the best and AAOBF follows. The weighted best-first algorithms have similar gap qualities. The trend is the same at longer time bounds.
- On `pedigree` with i-bound 18 at the 1 minute time bound, the average gap qualities of AAOBF, LAOBF, WAOBF, WAOBF-REP, and WRBFAOO are 0.133, 0.208, 0.286, 0.242, and 0.283, respectively. Here, AAOBF shows notable superiority compared to LAOBF, and the overall trend remains similar to the results from the other benchmarks.
- On `promedas` with i-bound 18 at the 1 minute time bound, the average gap qualities of AAOBF, LAOBF, WAOBF, WAOBF-REP, and WRBFAOO are 0.531, 0.496, 0.607, 0.577, and 0.562, respectively. Again, AAOBF and LAOBF perform better than the corresponding weighted best-first search algorithms.

In summary, we see that the hybrid AND/OR search algorithms have superior average gap quality compared to the weighted best-first AND/OR search algorithms across all benchmark domains considered.

6. Related Work

Solving marginal MAP exactly by depth-first branch and bound search was first introduced in Park and Darwiche (2003) who developed an unconstrained join-tree based upper bound to guide the

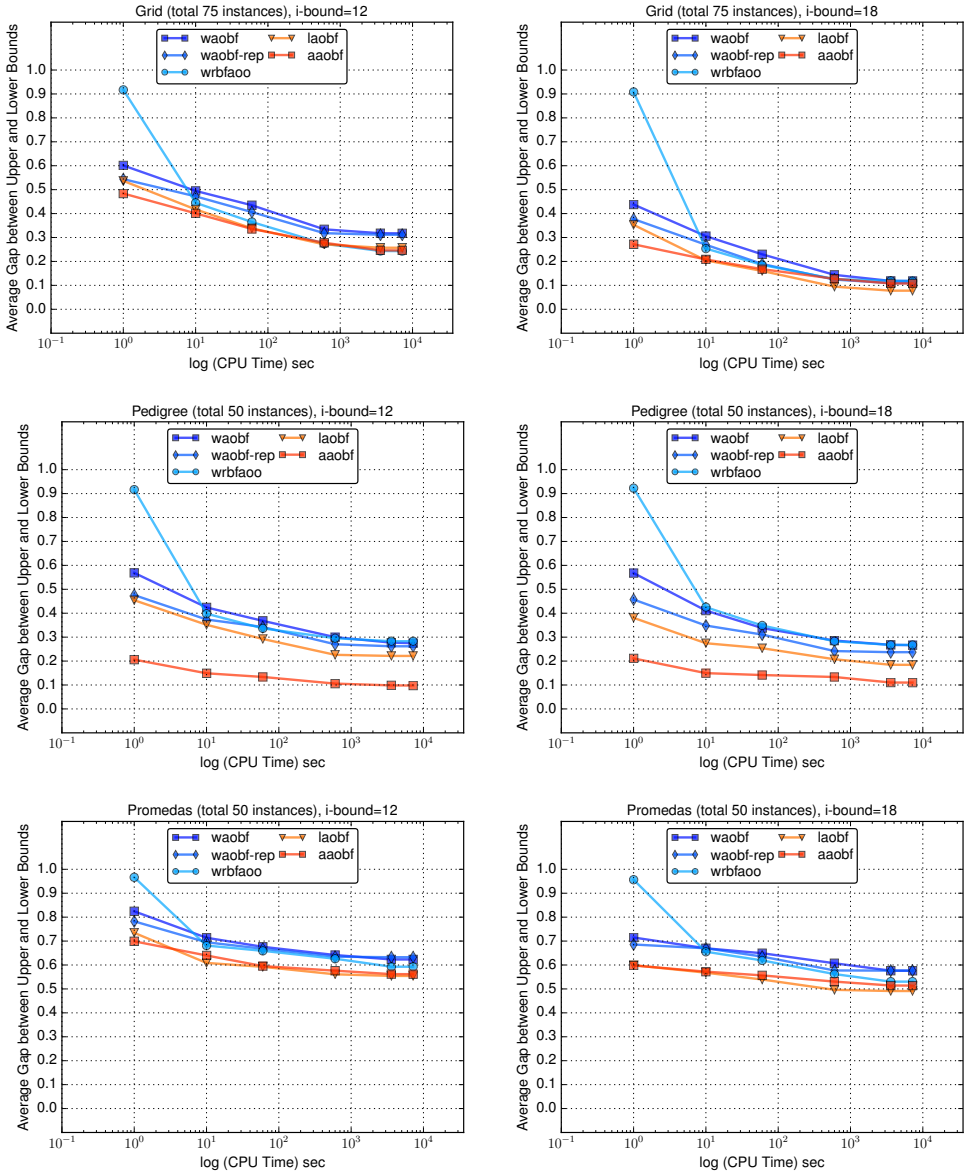


Figure 9: The average relative anytime gap qualities with the i-bound 12 (left) and 18 (right). Each line corresponds to a search algorithm that produce both the upper bounds and feasible solutions. The average gap between upper and lower bounds of all search algorithms decrease with time, and AAOBF produced high quality solutions quickly compared with the other algorithms.

search. The join-tree is fully re-valuated at each search node in order to compute upper bounds for all uninstantiated MAP variables simultaneously which allows for dynamic variable orderings. Subsequently, Yuan and Hansen (2009) proposed an incremental evaluation of the unconstrained

join-tree based upper bound which reduces significantly the computational overhead during search. However, this requires the search to be conducted along a static variable ordering.

Approximation algorithms for marginal MAP, including message passing and variational methods, have also been introduced (Liu & Ihler, 2013; Jiang, Rai, & Daume, 2011; Cheng, Chen, Dong, Xu, & Ihler, 2012; Ping et al., 2015). Upper and lower bound versions of these methods are closely related to the WMB heuristic we use to guide the search. However, unlike our search based algorithms, these schemes do not guarantee eventual optimality of their solutions in an anyspace way, i.e., without significantly increasing memory requirements.

The anytime marginal MAP algorithm introduced recently by Maua and Campos (2012) is another approach that can provide upper and lower bounds. It is an iterative join-tree based algorithm that propagates sets of messages (called factor sets) between the clusters of the join tree. These messages, however, tend to grow very large and therefore the method is limited to solving relatively easy problems with small induced widths.

The exact AND/OR search algorithms for marginal MAP was first introduced by (Marinescu et al., 2014) and the anytime AND/OR search by extending exact best first search by weighted best first search scheme was introduced by (Lee et al., 2016). The hybrid of depth-first and best-first AND/OR search algorithms that provides both upper and lower bounds was introduced by (Marinescu et al., 2017). This paper presents the above AND/OR search algorithms and related algorithms for marginal MAP with additional proofs and extended evaluations from coherent benchmark domains.

Hybrids of depth-first and best-first search The idea of combining depth-first and best-first search was first described in Pearl (1984). A similar idea was also employed in mixed integer programming algorithms as a plunging strategy in Achterberg (2007). The work that is closest in spirit to our LAOBF is that by Stern et al. (2010) who developed an A* with depth-first lookaheads in the context of generic path-finding. More recently, (Allouche, de Givry, Katsileros, Schiex, & Zytnicki, 2015) introduced a hybrid best-first search algorithm for solving Weighted CSPs that can be guided by a tree decomposition. It selects a node in the best-first manner and performs standard depth-first search with an adaptive number of backtracks to expand the frontier.

7. Conclusions

In this paper, we developed depth-first and best-first search algorithms 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 focused on the algorithms as exact solvers at first, and then extended them into anytime algorithms that produce improved solutions with time as well as tightened upper and lower bounds. Through extensive empirical evaluations on a variety of benchmarks we demonstrate the effectiveness of these algorithms as *exact* schemes against previous unconstrained join-tree based methods, which we also extend to be sensitive to high induced-width models. Our results show not only orders of magnitude improvements over the existing methods, but also the ability to optimally solve many instances that could not be solved before. In particular, the limited memory recursive best-first AND/OR search scheme consistently solved more problems within a two hour time-bound and in many cases was faster, by orders of magnitude, compared with all depth-first OR search counterparts. It emerges as the superior exact scheme overall.

Our proposed new anytime search algorithms for marginal MAP algorithms are based on either weighted search or interleaving best-first and depth-first search. These schemes are able to compute not only anytime lower bounds, but also anytime upper bounds on the optimal marginal MAP value,

and the gap between the two can be used to better gauge the solution quality during search. Our extensive empirical evaluation on various benchmarks demonstrates the effectiveness of the new algorithms compared with state-of-the-art depth-first search. We are able to show conclusively that our new best+depth-first search approach produces superior quality solutions more quickly than the pure best-first or depth-first search as well as the weighted schemes. They also give tighter gap guarantees compared to weighted best-first search.

The immediate question called for is how to extend our schemes to cases where the conditioned summation task is not tractable (remember that we addressed only cases when the number of MAP variables were large enough to bound the hardness of the summation task). So, for future work we plan to extend our proposed anytime search schemes to also handle effectively marginal MAP problems where the conditioned summation subproblem is not tractable anymore. We also plan to explore parallel search schemes for solving marginal MAP queries. Finally, we will extend to marginal MAP recent look-ahead and subproblem ordering ideas that proved effective for pure optimization tasks (Lam, Kask, Larrosa, & Dechter, 2017, 2018).

References

Achterberg, T. (2007). *Constraint Integer Programming*. Ph.D. thesis, Zuse Institute Berlin.

Allouche, D., de Givry, S., Katsileros, G., Schiex, T., & Zytnicki, M. (2015). Anytime hybrid best-first search with tree decomposition for weighted csp. In *International Conference on Principles and Practice of Constraint Programming*, pp. 12–29.

Chakrabarti, P., Ghose, S., & Sarkar, S. D. (1987). Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, *34*(1), 97–113.

Cheng, Q., Chen, F., Dong, J., Xu, W., & Ihler, A. (2012). Approximating the sum operation for marginal-map inference. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pp. 1882–1887. AAAI Press.

Cooper, G. (1990). The computational complexity of probabistic inferences. *Artificial Intelligence*, *42*, 393–405.

de Kleer, J., Mackworth, A. K., & Reiter, R. (1990). Characterizing diagnoses. In *Proceedings of the 8th National Conference on Artificial Intelligence. Boston, Massachusetts, July 29 - August 3, 1990, 2 Volumes.*, pp. 324–330.

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

Dechter, R., Kask, K., & Larrosa, J. (2001). A general scheme for multiple lower bound computation in constraint optimization. In *Principles and Practice of Constraint Programming*, pp. 346–360.

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

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

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

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

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

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

Flerova, N., Marinescu, R., & Dechter, R. (2017). Weighted heuristic anytime search: new schemes for optimization over graphical models. *Ann. Math. Artif. Intell.*, *79*(1-3), 77–128.

Freuder, E. C., & Quinn, M. J. (1985). Taking advantage of stable sets of variables in constraint satisfaction problems. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985*, pp. 1076–1078.

Geffner, H., & Bonet, B. (2013). *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Globerson, A., & Jaakkola, T. (2007). Approximate inference using conditional entropy decompositions. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 130–138.

Ihler, A., Flerova, N., Dechter, R., & Otten, L. (2012). Join-graph based cost-shifting schemes. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 397–406.

Jiang, J., Rai, P., & Daume, H. (2011). Message-passing for approximate map inference with latent variables. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., & Weinberger, K. Q. (Eds.), *Advances in Neural Information Processing Systems 24*, pp. 1197–1205. Curran Associates, Inc.

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

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

Lam, W., Kask, K., Larrosa, J., & Dechter, R. (2017). Residual-guided look-ahead in AND/OR search for graphical models. *J. Artif. Intell. Res.*, *60*, 287–346.

Lam, W., Kask, K., Larrosa, J., & Dechter, R. (2018). Subproblem ordering heuristics for AND/OR best-first search. *J. Comput. Syst. Sci.*, *94*, 41–62.

Lee, J., Marinescu, R., Dechter, R., & Ihler, A. (2016). From exact to anytime solutions for marginal MAP. In *30th AAAI Conference on Artificial Intelligence*, pp. 1749–1755.

Lee, J., Marinescu, R., & Dechter, R. (2014). Applying marginal map search to probabilistic conformant planning: Initial results.. In *AAAI Workshop: Statistical Relational Artificial Intelligence*.

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

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

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

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

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

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

Marinescu, R., Kask, K., & Dechter, R. (2003). Systematic vs non-systematic algorithms for solving the MPE task.. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 394–402.

Marinescu, R., Lee, J., Dechter, R., & Ihler, A. (2017). Anytime best+depth-first search for bounding marginal MAP. In *31th AAAI Conference on Artificial Intelligence*, pp. 1749–1755.

Mateescu, R., & Dechter, R. (2007). A comparison of time-space scheme for graphical models. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, pp. 2346–2352.

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

Mauá, D. D. (2016). Equivalences between maximum a posteriori inference in bayesian networks and maximum expected utility computation in influence diagrams. *Int. J. Approx. Reasoning*, *68*(C), 211–229.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Otten, L., & Dechter, R. (2011). Anytime AND/OR depth-first search for combinatorial optimization. In *International Symposium on Combinatorial Search*, pp. 117–702.

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

Park, J., & Darwiche, A. (2004). Complexity results and approximation strategies for MAP explanations. *Journal of Artificial Intelligence Research*, *21*(1), 101–133.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies*. Addison-Wesley.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Ping, W., Liu, Q., & Ihler, A. T. (2015). Decomposition bounds for marginal MAP. In *Advances in Neural Information Processing Systems 28*, pp. 3267–3275.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, *1*(3-4), 193–204.

Shimony, S., & Charniak, E. (1991). A new algorithm for finding MAP assignments to belief networks. In *P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer (Eds.), Uncertainty in Artificial Intelligence*, Vol. 6, pp. 185–193.

Stern, R., Kulberis, T., Felner, A., & Holte, R. (2010). Using lookahead with optimal best-first search. In *24th AAAI Conference on Artificial Intelligence*, pp. 185–190.

Wainwright, M., Jaakkola, T., & Willsky, A. (2005). A new class of upper bounds on the log partition function. *IEEE Trans. Info. Theory*, *51*(7), 2313–2335.

Weiss, Y., Yanover, C., & Meltzer, T. (2007). MAP estimation, linear programming and belief propagation with convex free energies. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 416–425.

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