Heuristic Search for m Best Solutions with Applications to Graphical Models

Rina Dechter and Natalia Flerova

Bren school of Information and Computer Sciences
University of California Irvine

Abstract. The paper focuses on finding the m best solutions to a combinatorial optimization problems using Best-First or Branch-and-Bound search. We are interested in graphical model optimization tasks (e.g., Weighted CSP), which can be formulated as finding the m-best solution-paths in a weighted search graph. Specifically, we present *m*-A*, extending the well-known A* to the m-best problem, and prove that all A*'s properties are maintained, including soundness and completeness of *m-A**, dominance with respect to improved heuristics and most significantly optimal efficiency compared with any other search algorithm that use the same heuristic function. We also present and analyse *m-B&B*, an extension of a Depth First Branch and Bound algorithm to the task of finding the *m* best solutions. Finally, for graphical models, a hybrid of A* and a variable-elimination scheme yields an algorithm which has the best complexity bound compared with earlier known m-best algorithms.

1 Introduction

Depth-First Branch and Bound (*B&B*) and Best-First Search (BFS) are the most widely used search schemes for finding optimal solutions. In this paper we explore the extension of such search algorithms to finding the m-best optimal solutions. We apply such algorithms to optimization tasks over graphical models, such as weighted CSPs and most probable explanation (MPE) over probabilistic networks, arising in many applications, e.g, in procurement auction problems, biological sequence alignment and finding m most likely haplotype configurations.

Most of the paper's analysis focuses on Best-First Search whose behavior for the task of finding a single optimal solution is well understood. The algorithm is known to be sound and complete when guided by an admissible (i.e., lower bound for minimization task) heuristic evaluation function. Most significantly, it is efficiently optimal: any node it expands must be expanded by any other exact search algorithm having the same heuristic function, if both use the same tie breaking rule [4]. Best-First Search, and its most famous variant A*, are also known to require significant memory. The most popular alternative to BFS is Depth-First Branch and Bound, whose most attractive feature compared with BFS is that it can be executed with linear memory. Yet, when the search space is a graph, it can exploit memory to improve its performance, by flexibly trading space and time.

Highly efficient $B\&B$ and BFS algorithms for finding an optimal solution over graphical models were developed recently. These algorithms explore the AND/OR search-tree or the context-minimal AND/OR search graph of the graphical model [3], they use heuristic evaluation functions generated either by the mini-bucket scheme or through soft arc-consistency schemes [9, 12] and they proved to be most effective as demonstrated in recent evaluations [2]. Clearly, an extension of a BFS or $B\&B$ to the m-best task over a general search space graph for optimal path-finding tasks is applicable to graphical models when searching its AND/OR search space [3].

The main contribution of this paper (Section 3) is in presenting m-A*, an extension of the A* for finding the m-best solutions, and in showing that all its properties extend to the m-best case. In particular we prove that m-A* is sound and complete and is efficiently optimal. In Section 4 we discuss extensions of Branch and Bound to the m-best task. Subsequently, in Section 5, we discuss anchoring the algorithms for searching graphical models' AND/OR search spaces. We show that a hybrid of A* and a variable-elimination scheme, denoted BE-Greedy-m-BF, yields a best complexity algorithm compared with earlier work on graphical models. Preliminary empirical evaluation shed light on the algorithm's performance. We assume without loss of generality that the optimization task at hand is minimization.

2 Background

Let A be a general search algorithm for finding an optimal solution path over a search space defined implicitly by a set of states (the nodes in the graph), operators that map states to states having costs or weights (the directed weighted arcs), a starting state n_0 and a set of goal states. The task is to find the least cost solution path from the start node to a goal [10] where the cost of a solution path is the sum of the weights or the product of the weights on its arcs. The two primary search strategies for finding an optimal solution are Best-First Search (BFS), (e.g., A*) and Depth-First Branch-and-Bound search ($B\&B$). In this paper we explore extensions of both schemes for finding the m-best solutions for any integer m. Best-first search (BFS) seems to be the most suitable algorithm to be extended to m-best task. It explores the search space using a heuristic evaluation function $f(n)$ which for every node n estimates the best cost solution path passing through n. The algorithm maintains a list of nodes that are candidates for expansions (often called "OPEN", or "frontier"). At each iteration a node in the frontier having a minimal f is selected for expansion, moved to another list (called "CLOSED", or "explored set") and its child nodes are places in OPEN, each associated with its own evaluation function. When a goal node is selected for expansion the algorithm terminates and outputs the solution found.

It is known that when $f(n)$ is a lower bound on the optimal cost path that goes through n the algorithm terminates with an optimal solution. The algorithm's properties were extensively studied [10, 11]. In particular, (up to some tie breaking rule) the algorithm is efficiently optimal. Namely, any node expanded by BFS is expanded by any other search algorithm guaranteed to find an optimal solution [4]. If provided with a consistent (also called monotonic) heuristic, the algorithm expands nodes in frontiers of monotonically increasing evaluation function and it expands every node just once [10]. This later property is of utmost importance because it frees the algorithm from the need to check for duplicates when the search space is a graph.

We focus on the well-known BFS variant called A^*, where the heuristic evaluation function is expressed as the sum $f(n) = g(n) + h(n)$ in which for any node n $g(n)$ is minimal cost from the root n_0 to n along the current path, and $h(n)$ underestimates $h^*(n)$, the optimal cost from n to a goal node. The implicit directed search space graph is $G = (N, E)$. We denote by $g_\pi(n)$ the cost from the root to n along path π and by $c_\pi(n_1, n_2)$ the cost from n_1 to n_2 along π. The heuristic function h is *consistent* iff $\forall$ n' successor of n in G, $h(n) \leq c(n, n') + h(n')$.

3 Best-first Search For the Best m Solutions

As was noted in [1], the extension of BFS algorithms, including A^*, to the m-best problem seems simple: rather then terminate with the first solution found, the algorithm continues searching until it generates m solutions. As we will show, these solutions are the m best ones. Specifically, the second solution found is the second optimal solution, the third is the third optimal one and so on. In the following subsections we present an extension of A^* to finding the m-best solutions and show that most of A^*'s nice properties are maintained in m-A^*.

3.1 Algorithm m-A*

Figure 1 provides a high level description of a tree-search variant which we call m-A*. The algorithm expands nodes in order of increasing value of f in the usual A^* manner. For simplicity we specify the algorithm under the assumption that h is consistent. The algorithm maintains separate paths to each copy of a node in the explored search tree, denoted by Tr. As we will show, this redundancy is not wasteful when the heuristic function is consistent.

We denote by C^*_i the i^{th} best solution cost, by $f^*_i(n)$ the cost of the i^{th} best solution going through node n, by $f_i(n)$ the evaluation function estimating $f^*_i(n)$ and by $g_i(n)$ and $h_i(n)$ the estimates of the i^{th} best costs from n_0 to n and from n to a goal, respectively. If the heuristic function is not consistent, step 7 should be revised to account for the possibility that new paths to n' may be encountered and expanded in a non-monotone cost order. We therefore should revise as follows:
7a. for any node n' that appears already more than m times in the union of OPEN or CLOSED, if $g(n')$ is strictly smaller than $g_m(n')$, the current m-best path to n', keep n' with a pointer to n and discard the earlier pointer.

Algorithm m-A*
Input: An implicit directed search graph $G = (N, E)$, with a start node n_0 and a set of goal nodes *Goals*. A consistent heuristic evaluation function $h(n)$, parameter m, OPEN=$\emptyset$. A tree Tr which is initially empty.
Output: the m-best solutions.

```
1. i =1 (i counts the current solution being searched for).
2. Put the start node $n_0$ in OPEN, $f(n_0) = h(n_0)$. Make $n_0$ the root of $Tr$.
3. If OPEN is empty, exit and return the solutions found so far.
4. Remove a node, denoted $n$, in OPEN having a minimum $f$ (break ties arbitrarily,
   but in favor of goal nodes and deeper nodes) and put it in CLOSED.
5. If $n$ is a goal node, output the current solution obtained by tracing back pointers
   from $n$ to $n_0$ (pointers are assigned in the following step). Denote this solution as
   $Sol_i$. If $i = m$ exit. else $i \leftarrow i + 1$, and go to step 3.
6. Otherwise expand $n$, generating all its child nodes $Ch$. Compute $g(n') = g(n) +
   c(n, n')$ and $f(n') = g(n') + h(n')$, $\forall n' \in Ch$.
7. If $n'$ appears in OPEN or CLOSED $m$ times, discard node $n'$, otherwise attach
   from each $n'$ in $Ch$ a pointer back to $n$ in $Tr$. Insert $n'$ into the right place in
   OPEN based on its $f(n')$.
8. Go to step 3.
```

Fig. 1: Algorithm m-A*

Example 1. Consider example in Figure 2. We assume the task of finding the two shortest paths from node A to node G. Assuming the following heuristic values: h(A)=5, h(B)=4, h(C)=3, h(D)=2, h(E)=1, h(F)=1, h(G)=0, the cost of the best solution path $\{A, C, D, F, G\}$ is cost 6, the cost of the second best solution $\{A, B, D, F, G\}$ is cost 9. On the trace of the search tree (right) orange nodes were expanded and put on CLOSED, blue nodes remain on OPEN, magenta nodes are the goals. Nodes D, F, E and G are expanded twice and two duplicates of each of these nodes are retained.

Based on the known properties of A^* [10] we will establish the following corresponding properties of *m-A**:

1. Soundness and completeness: *m-A** terminates with the m-best solutions generated in order of their costs.
2. Optimal efficiency: *m-A** is optimally efficient in terms of node expansions, compared with any other search algorithm that is guaranteed to find the m-best solutions. Namely, any node that is surely expanded by *m-A** must be expanded by any other sound and complete algorithm.
3. Optimal efficiency for consistent heuristics: *m-A** is optimally efficient in terms of *number of node expansions* when the heuristic function is consistent. In this case, *m-A** expands each node at most m times.
4. Dominance: Given two heuristic functions h_1 and h_2, s.t. for every n $h_1(n) < h_2(n)$, *m-A**$_1$ will expand every node surely expanded by *m-A**$_2$, when *m-A**$_i$ is using heuristic h_i.

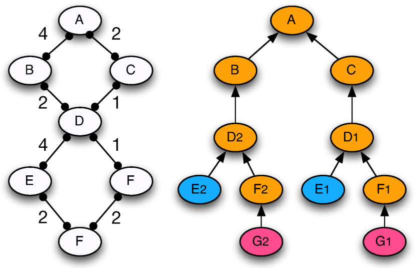


Fig. 2: The search graph of the problem (left) and the trace of the m-A^* for m=2. Node A is the start node, node G is the goal node. Orange represents the nodes that were expanded and put on CLOSED, blue nodes were generated, but never expanded (remain on OPEN list), magenta nodes are the goal nodes.

3.2 *m-A** is Sound and Complete

We know, that, if provided with an admissible heuristic, A^* will surely expand any node n, whose evaluation function f_π along a given path π from n_0 to n is strictly lower than the cost of the optimal solution, C^*. Namely, it surely expands every node n', such that $n' \in \pi_{n_0..n}$ and $f(n') < C^*$. We next show that this property can be extended straightforwardly to the m-best case.

Let's denote by SS_i the set of nodes expanded by *m-A** just before a goal node of the i^{th}-best solution was selected for expansion. By definition $SS_1 \subset SS_2, ... \subset SS_i, .. \subset SS_m$ and $C_1^* \leq C_2^*, ... \leq C_m^*$.

Since we maintain at most m live copies of a node in OPEN or CLOSED we must be sure that we never discard any of the viable m-best solution paths. Bounding the number of copies of a node is important for complexity reasons.

Proposition 1. *At any point before the algorithm generates the i^{th}-best solution there is always a node in OPEN along each of the j^{th}-best solution path for $j \in [i, ..., m]$.*

From the above proposition it follows that,

Proposition 2. *At any time before m-A* expands a goal node of the i^{th} best solution path, there is a node n' in OPEN satisfying $f(n') \leq C_i^*$.*

Proof. Assume that the search graph G has at least m solutions. Let's assume that all the $i-1$ best solutions were already generated. Namely, the goal nodes of the first $i-1$ solution paths were selected for expansion. At any time after that, there is a node n in OPEN that resides on an i^{th} best path π_i, because the i^{th}-best path is not discarded (Proposition 1). Let n' be the first OPEN node on π_i. Its evaluation function is, by definition, $f(n') = g_{\pi_i}(n') + h(n')$. Since π_i is an i^{th} best path, $g_{\pi_i}(n') + c_{\pi_i}(n', t) = C_i^*$, when t is the goal for path π_i. Since by definition $h(n') \leq c_\pi(n', t)$, we get that n' is in OPEN and $f(n') \leq C_i^*$.

We can conclude:

Theorem 1 (sound and completeness). *Algorithm m-A* generates the m-best solutions in order, namely, the i^{th} solution generated is the i^{th} best solution.*

Proof. By induction. We know that the first solution generated is the optimal one and, assuming that the first $i-1$ solutions generated are $i-1$ best in sequence, we will prove that the i^{th} one generated is the i^{th} best. If not, the i^{th} generated solution path, denoted by π' has a cost c and $c > C_i^*$. However, when the algorithm selected the goal t' along π', its evaluation function was $f(t') = g_{\pi'}(t') = c$, while there was a node in OPEN whose evaluation function is C_i^* or smaller that should have been selected. Contradiction.

3.3 *m-A** is Optimally Efficient

Algorithm A* is known to be optimally efficient [4]. Namely, any other algorithm that extends search paths from the root and uses the same heuristic information will expand every node that is surely expanded by A^*, i.e., $\forall\ n$, such that $f(n) < C^*$. This property can be extended to our *m-A** as follows:

Theorem 2 (m-optimal efficiency). *Any search algorithm which is guaranteed to find the m-best solutions, and which explores the same search graph as m-A* will have to expand any node that is surely expanded by* m-A**, if it uses the same heuristic function. Formally, it will have to expand every node n that lies on a path $\pi_{0..n}$ that is dominated by C_m^*, namely s.t., $f(n') < C_m^*\ \forall n' \in \pi_{0..n}$.*

Similarly to [4] we can show that any algorithm that does not expand a node n lying on a path $\pi_{0..n}$, whose evaluation function is dominated by C_m^*, (namely $\forall\ n'\ f(n') < C_m^*$), can miss one of the m-best solutions when applied to a slightly different problem, and therefore contradicts completeness.

Proof. Consider a problem having the search graph G and consistent heuristic h. Assume that node n is surely expanded by *m-A**, namely for some $j \leq m$ node n lies on a path π dominated by C_j^*. Let B be an algorithm that uses the same heuristic h and is guaranteed to find the m best solutions. Assume that B does not expand n. We can create a new problem graph G' (Figure 3) by adding a new goal node t with $h(t) = 0$, connecting it to n by an edge having cost $c = h(n) + \delta$, where $\delta = 0.5(C_j^* - D)$, in where $D = \underset{n' \in E_j}{max} f(n')$, E_j is the set of nodes surely expanded by *m-A** before finding the j^{th} solution.

It is possible to show that the heuristic h is also admissible for the graph G' [4]. Since $\delta = 0.5(C_j^* - D)$, $C* = D - 2\delta$. By construction, the evaluation function of the new goal node is $f(t) = g(t) + h(t) = g(n) + c = g(n) + h(n) + \delta = f(n) + \delta \leq D + \delta = C_j^* - \delta < C_j^*$. Since *m-A** will surely expand all nodes with $f(n') < C_j^*$ before finding the j^{th} solution, it will expand node t and discover the solution whose cost is bounded by $C_j^* - \delta$. On the other hand, algorithm B, that does not expand node n in the original problem, will be unable to reach node t and will only discover the solution with cost C_j^*, thus not returning the true set of m best solutions to the modified problem. Contradiction.

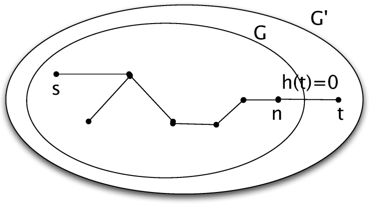


Fig. 3: The graph G' represents a new problem instance constructed by appending to node n a branch leading to a new goal node t.

3.4 *m-A** for Consistent Heuristics

If the heuristic function is consistent, whenever a node n is selected for expansion (for the first time) by A^* the algorithm had already found the shortest path to that node. We can extend this property as follows:

Theorem 3. *Given a consistent h, when* m-A* *selects a node n for expansion for the i^{th} time, then $g(n) = g_i^*(n)$, namely it has found the i^{th} best solution from s to n.*

Proof. By induction. For $i = 1$ the theorem holds [10]. Assume that it also holds for $i = (j - 1)$. Let us consider the j^{th} expansion of n. Since we have already expanded n $(j - 1)$ times and since by the induction hypothesis we have already found the $(j - 1)$ distinct best paths to the node n, there can be two cases: either the cost of the j^{th} path to n is equal to the j^{th} best, i.e., $g(n) = g_i^*(n)$, or it is greater i.e., $g(n) = g_k^*(n), k > i$. If we assume the latter, then there exists some other path π from n_0 to n with cost $g_\pi(n) = g_j^*(n) < g_k(n)$ that has not been traversed yet. Since $f_\pi(n) = g_\pi(n) + h(n)$ and $f(n) = g(n) + h(n)$, it follows that $f_\pi(n) < f(n)$. Let n' be the latest node on π on OPEN, i.e., a node already generated but not yet expanded. It is known, that if the heuristic function is consistent, the values of f along any given path are non-decreasing and therefore $\forall\, n'$ on π, $f_\pi(n') \leq f_\pi(n) < f(n)$ and n' should have been expanded before node n leading to contradiction.

We can conclude that when h is consistent any node n will be expanded at most m times.

Corollary 1. *Given m-A* with a consistent h*

- *The maximum number of copies of the same node in OPEN or CLOSED can be bounded by m.*
- *The set $\{n | f(n) < C_m^*\}$ will surely be expanded (no need of dominance along the path).*

3.5 The Impact of m on the Expanded Search Space

The relation between the sizes of search space explored by m-A* for different levels of m is obviously monotonically increasing with m. We can provide the following characterization for the respective search spaces.

Proposition 3. *Given a search graph,*

1. *Any node expanded by i-A* is expanded by j-A* if $i < j$ and if both use the same tie-breaking rule.*
2. *The set $S(i,j)$ of nodes defined by $S(i,j) = \{n | C_i^* < f(n) < C_j^*\}$ will surely be expanded by j-A* and surely not expanded by i-A*.*
3. *If $C_i = C_i^*$, the number of nodes expanded is determined by the tie-breaking rule.*

As a result, the larger the discrepancy between the respective costs $C_j^* - C_i^*$ is, the larger would be the potential difference in the search spaces they explore. This, however, also depends on the granularity with which the values of a sequence of observed evaluation functions increase, which is related to the arc costs (or weights) of the search graph. If $C_i^* = C_j^* = C$, then the search space explored by i-A* and j-A* will be different only in the frontier of $f(n) = C$.

3.6 The Case of $h = h^*$ for m-A*

Like A^*, *m-A** improves its performance if it has access to more accurate heuristics. In particular, when h_1 is strictly larger (and therefore more accurate) than h_2, every node surely expanded by *m-A** with h_2 before the j^{th} solution is uncovered will also be expanded by *m-A** with h_1 before the j^{th} solution is uncovered. The proof idea is identical to the A* case and is omitted. The case of the exact heuristic deserves a special notice. It is easy to show that,

Theorem 4. *If $h = h^*$ is the exact heuristic, then m-A* generates solutions on j-optimal paths $1 \leq j \leq m$, only.*

Proof. Since the heuristic function is exact the f values in OPEN are expanded in sequence $C_1^* \leq C_2^* \leq ... \leq C_i^* ... \leq C_m^*$. All the nodes generated having $f = C_1^*$ are by definition on optimal paths (since $h = h^*$), all those generated who have $f = C_2^*$ must be on paths that can be second best and so on. (Notice that the best cost may differ only at some indices.)

When $h = h^*$ m-A* is clearly linear in the number of nodes having $f^* \leq C_m^*$ value. However, when the cost function has only a small range of values, there may be exponential number of solution paths having cost C_m^*. To avoid this exponential frontier we chose the tie-breaking rule of expanding deeper nodes first. This will result in a number of nodes expansions that is bounded by mn when n bounds the solution length.

In summary,

Theorem 5. *When m-A* has access to $h = h^*$, then, if it uses a tie breaking rule in favor of deeper nodes, it will expand at most $\#N$ nodes, where $\#N = \sum_i \#N_i$ and $\#N_i$ is the length of the i-optimal solution path. Clearly $\#N \leq mn$.*

4 Branch and Bound for m-best Solutions

Straightforwardly extending the well-known *B&B* scheme, algorithm m-Branch-and-Bound (*m-B&B*) finds the m best solutions by exploring the search space in a depth first manner.

The algorithm maintains an ordered list of the m best solutions found so far. It prunes nodes whose evaluation function (a lower bound on the optimal solution passing through the node) is greater than U_m, current upper bound on the m^{th} best solution ($U_m = \infty$ until initial m solutions are found). At time of completion, *m-B&B* outputs an ordered list of the m best solutions.

In analyzing the complexity of *m-B&B* we assume that the underlying search space is a graph (as opposed to a search tree). The three main sources of complexity, compared with finding a single best solution, are: the expansion of the search space, the node processing overhead due to m best problem and the impact of the cost function on pruning the explored search space. We defer the analysis of the first two factors till Section 5.3, where we discuss the application of *m-B&B* to graphical models.

Similar to m-A*, the number of nodes expanded by *m-B&B* greatly depends on the values of the evaluation function and costs of the solutions. Since, as we have already shown, m-A* is superior to any exact search algorithm for m-best solutions, *m-B&B must* expand all the nodes that are *surely expanded* by m-A*, which, assuming consistent heuristic, is the set of nodes $E^*_m = \{n|f(n) < C^*_m\}$.

The order in which *m-B&B* uncovers solutions is problem-specific and has a big impact on the total number of node expantions. Let $\{U^1_m, ..., U^j_m\}$ be the sequence of upper bounds on the m^{th} best solution, at the time when *m-B&B* uncovered j solutions, ordered from the earliest bound obtained to the latest. These upper-bounds, that influence the node pruning, are decreasing until they coincide with C^*_m.

Proposition 4. *Given a consistent heuristic*

1. m-B&B *will expand all nodes such as* $E^*_m = \{n|f(n) < C^*_m\}$.

2. Initially, and until m-B&B *encounters the true* m^{th}*-best solution, it will expand also nodes* $F^*_m = \{n|f(n) > C^*_m\}$. *Subsequently, like m-A* it will only expand nodes for which* $f(n) \leq C^*_m$.

5 Applying m-A* and *m-B&B* to Graphical Models.

In this section we apply m-A* and *m-B&B* to combinatorial optimization tasks (COP) defined over graphical models. We focus more on the application of *m-B&B*, since it is more practical then m-A* due to flexible memory requirements.

5.1 Background on graphical models.

Consider a weighted constraint problem expressed as a **graphical model** $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \sum)$, where $\mathbf{F} = \{f_1, \ldots, f_r\}$ is a set of discrete functions, called costs, over real-valued variables $\mathbf{X} = \{X_1, \ldots, X_n\}$ with discrete domains $\mathbf{D} = \{D_1, \ldots, D_n\}$. We aim to minimize the sum of all costs $\min_{\mathbf{X}} \sum_i f_i$. Closely related combinatorial optimization problem is Most Probable Explanation (MPE), where the task is to compute $\max_{\mathbf{X}} \prod_i f_i$, where the sum is replaced by a product. The set of function scopes implies a *primal graph* and, given an ordering of the variables, an *induced graph* (where, from last to first, each node's earlier neighbors are connected) with a certain *induced width* w^*.

Example 2. Figure 4a depicts the primal graph of a problem with six variables. Figure 4b shows its induced graph along the ordering $d = A, B, C, D, E, F$, $w^* = 2$.

AND/OR search space of a graphical model exploits the problem decomposition captured in the structure of the graph (see [3] for more details). The search space is defined using a **pseudo tree** of the graphical model. A pseudo tree of an undirected graph $G = (X, E)$ is a directed, rooted tree $\mathcal{T} = (X, E')$, such that every arc of G not included in E' is a back-arc in $\mathcal{T}$, namely it connects a node in T to an ancestor in $\mathcal{T}$. The arcs in E' may not all be included in E.

AND/OR Search Trees. Given a graphical model $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \sum)$ and a pseudo tree $\mathcal{T}$, the *AND/OR search tree* guided by $\mathcal{T}$ consists of alternating levels of OR and AND nodes. Figure 4d shows the AND/OR search tree for our running example, guided by the pseudo-tree in Figure 4c.

It was shown that, for a problem with N variables with domain size k and a pseudo tree $\mathcal{T}$ of height h, the size of the corresponding AND/OR search tree is $O(Nk^h)$. Alternatively the size can be bounded by $O(Nk^{w^* \log N})$, where w^* is the induced width of the problem graph along a depth-first traversal of $\mathcal{T}$ [3].

AND/OR Search Graphs. Identical subproblems, identified by their context (the partial instantiation that separates the subproblem from the rest of the network), can be merged, yielding the *context-minimal AND/OR search graph*, at the expense of using additional memory during search. Merging nodes of the AND/OR tree in Figure 4d yields the context-minimal AND/OR search graph in Figure 4e. The context-minimal AND/OR search graph has size $O(Nk^{w^*+1})$ (see [3] for details).

An optimization problem can be solved by searching the corresponding *weighted AND/OR search graph*, namely an AND/OR search graph, in which each edge from OR node to AND node has a weight derived from the set of cost functions F [3]. As a heuristic evaluation function these search algorithms use the mini-bucket heuristic that is known to be admissible and consistent, yielding algorithms AOBB (AND/OR Branch and Bound) and AOBF (AND/OR Best-First), described and extensively studied in [7, 9]. The solution to the problem is a subtree of the weighted AND/OR search graph.

5.2 Algorithm m-AOBF for graphical models.

As noted, *AOBF* (AND/OR Best First) is version of algorithm A^* that searches the weighted AND/OR context minimal graph in a best-first manner. It can be guided by any admissible heuristic. Extending AOBF to the m-best task as suggested by the general m-A* yields an algorithm which we call *m-AOBF*. All

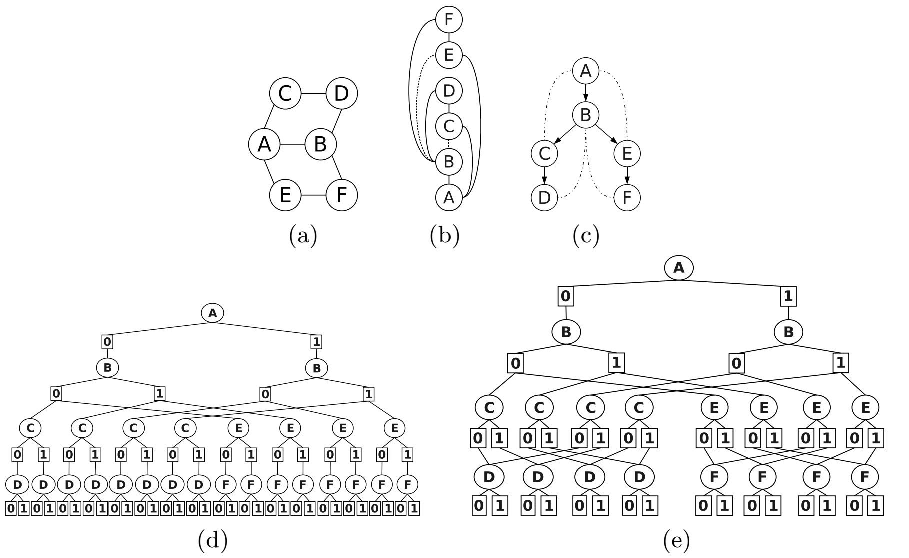


Fig. 4: Example primal graph with six variables (a), its induced graph along ordering $d = A, B, C, D, E, F$ (b), a corresponding pseudo tree (c), the resulting AND/OR search tree (d), and the context-minimal AND/OR search graph (e).

the properties of *m-A** extends to AND/OR search graphs. The algorithm may require duplicating some nodes at most m times (if there are m paths leading to them).

The complexity of m-best algorithms for graphical models can be analyzed by first characterizing the underlying search space that is being explored using graph parameters, as is common in graphical models. Subsequently, we can characterize the portion of the search space that is explored using the evaluation function, as we did earlier. So, since *m-AOBF* explores the AND/OR context-minimal graph (denoted CMG), and since we may duplicate some nodes at most m times we get:

Theorem 6. *The search space size explored by* m-AOBF *is bounded by* $O(N \cdot m \cdot k^{w*})$*, where* w^* *is the induced-width along the pseudo-tree ordering.*

Obviously this bound is loose since it does not consider the pruning power of the evaluation function.

Theorem 7. *Given an evaluation function* $f(n)$ *that underestimates the least cost solution tree that passes through the node n in the AND/OR context-minimal graph, algorithm* m-AOBF *surely expands the set of nodes* $\{n \in CMG | f(n) < C^*_m\}$*, where* C^*_m *is the cost of the* m^{th} *best solution, and a subset of nodes for which* $\{n \in CMG | f(n) = C^*_m\}$*, depending on the tie-breaking rule.*

5.3 Algorithm m-AOBB for Graphical models

Extending *AOBB* to *m-AOBB* is straightforward, mimicking the *m-B&B* while searching the AND/OR search space. The main difference between *m-AOBB* and *m-B&B* arises from the presence of AND nodes: at each AND node the m best solutions to the subproblems rooted in its children need to be combined and the best m out of the combination results need to be chosen. We distinguish between *m-AOBB* searching the AND/OR tree and the context minimal AND/OR graph. These two versions of *m-AOBB* differ in the size of underlying search space and the computational overhead per node.

The size of the AND/OR search tree is bounded by $O(Nk^h)$, where h is the height of the pseudo-tree. The primary overhead is due to the combination of the solutions to the children's subproblems of AND nodes which is $O(deg \cdot m \log m)$ time per AND node. Consequently, if the algorithm that searches the underlying AND/OR tree (with no caching) the run-time complexity is bounded by $O(m \cdot Nk^h \cdot deg \log m)$.

If the algorithm searches the context minimal AND/OR graph, (where identical subproblems are identified based on their context) *m-AOBB* need to cache the m best solutions rooted in some OR nodes, which requires additional memory, similar to graph-based AOBB [9]. Such caching introduces time overhead of $O(m \log m)$ per OR node. Since the size of an AND/OR search graph is bounded by $O(Nk^{w*})$ nodes and considering the AND node-related overhead, we can conclude that

Theorem 8. *The time complexity of m-AOBB which searches the underlying AND/OR search tree is bounded by $O(m \cdot Nk^h \cdot deg \log m)$. The time complexity of m-AOBB which searches the context-minimal graph is $O(N \cdot deg \cdot m \log mk^{w^*})$ and the space complexity is $O(N \cdot m \log mk^{w^*})$, where w^* is the induced-width of the ordering along the pseudo-tree that guides the search and h is its height.*

The above analysis considers only the impact of the m-best exploration on the size of the underlying search space and the overhead computation per node, but ignore the pruning power caused by the evaluation function that guides the search. Clearly the complexity analysis can be further refined by taking into consideration cost function, directly extending the results of Proposition 4 to *m-AOBB*.

5.4 Algorithm BE-Greedy-m-BF

Since an exact heuristic for graphical models can be generated by the bucket-elimination (BE) algorithm [7], we can use the idea suggested in Section 3.6, yielding algorithm which we call *BE-Greedy-m-BF*. The algorithm first generates the exact heuristics along an ordering using *BE* and subsequently applies *m-AOBF* using these exact heuristics. It turns out that the worst-case complexity of both algorithms when applied in sequence coincides with the best of the known m-best algorithms for graphical models. This is because the number of nodes expanded by m-A* when it uses the exact heuristic is bounded by $N \cdot m$, when N is the number of variables.

Theorem 9. *Let $B = (X, D, F)$ be a general graphical model. The complexity of* BE-Greedy-m-BF *(i.e., bucket-elimination followed by m-$AOBF$ with exact heuristic) is $O(Nk^{w*} + N \cdot m)$ when N is the number of variables.*

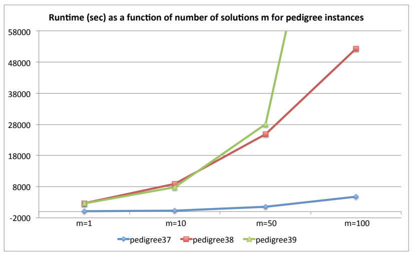
6 Earlier work on finding the m-best solutions.

The most influential work on finding the m-best solution was written by Lawler [8]. In the folowing years a great number of related methods were developed, which we do not describe here for lack of space. Particular areas of interest include the problem of finding k shortest paths, extensive references for which can be found in [5]. An overview of algorithms focused on graphical model is presented in [6].

7 Empirical demostrations

Instance	N	k	w*	h	m=1	m=10	m=50	m=100
pedigree1	298	4	15	59	1	32	464	1751
pedigree37	726	5	20	72	41	240	1537	4825
pedigree38	581	5	16	52	2700	8890	24894	t/o
pedigree39	953	5	20	77	2594	7907	28037	t/o
pedigree50	478	6	16	54	685	2835	21903	t/o
grid50-12-5	143	2	15	48	1	2	13	35
grid50-14-5	195	2	18	64	87	599	1995	3739
grid50-15-5	224	2	19	76	180	1382	5011	9591
grid75-16-5	256	2	21	73	539	3327	10917	20572
grid75-18-5	324	2	24	85	1147	4533	14078	23948
mm-03-0000	1220	2	18	43	43	580	4845	13381
mm-03-0001	1193	2	15	38	1	2	8	23
mm-03-0002	1193	2	15	39	1	3	17	48
mm-03-0004	1193	2	15	38	0	3	17	46
mm-03-0005	1193	2	15	38	0	5	25	70
mm-03-0011	1172	2	18	42	18	148	757	2016
mm-03-0014	1172	2	18	43	15	107	544	1488
mm-04-0012	2224	2	29	56	842	6859	31952	t/o
mm-04-0013	2224	2	29	58	375	4880	23854	t/o
mm-04-0014	2224	2	29	60	655	8080	36740	t/o
mm-04-0015	2224	2	29	57	818	13299	t/o	t/o

(a) Run-time of m-$AOBB$ in seconds.



(b) Pedigree instances. Pedigree 39 could not be solved before the time-out of 12 hours, thus the runtime for m=100 is unavailable.

Fig. 5: The run-time of m-$AOBB$ in seconds as a function of number of solutions m

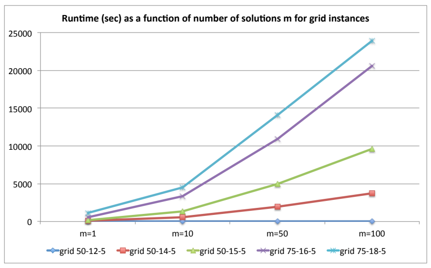
Due to our interest in extremely memory-intensive problems (e.g. genetic linkage analysis), we deem Branch and Bound-based methods with their flexible memory requirements the more promising in practice. In the following preliminary experiments we focused on evaluating how m-$AOBB$ scales with number of solutions m.

Our implementation of m-$AOBB$ is exploring an AND/OR search tree and not a graph. By making such design choice not only we achieve better space complexity, it also allows us to avoid the time overhead due to caching. On the other hand, the search space we are exploring is inheritently larger (see Section 5.3).

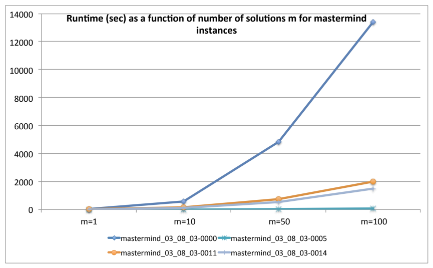
We evaluated the algorithm on maximum probability explanations (MPE) problems using 3 sets of instances taken from 2008 UAI evaluation: pedigrees, grids and mastermind instances. The parameters of the problems and run time results in seconds are presented in Table 5a.

We evaluated the algorithm for $m = [1, 10, 50, 100]$ solutions. The timeout was set to 12 hours. The memory limit for pedigree instances is 4 Gb, for grids and mastermind instances it is 10 Gb. In Figures 5b, 6a and 6b we see the dependence of the runtime on number of solutions m for a number of chosen instances from each set.

The theory states that the run-time of *m-AOBB* should scale with the number of solutions m as $(m \log_2 m)$. In practice, we see that it is not always the case and there exists a large discrepancy between the results. For some instances (e.g. grid 50-12-5, mm-03-001) the scaling of runtime with m is significantly better than what theory suggests, which can be attributed to sucessful pruning of a large part of the search space. However, for some instances (e.g. pedigree38, mm-04-0015) the runtime scales considerably worse. We explain the large scaling factor by the suboptimality of our current implementation, which might introduce overhead unaccounted for by theoretical analysis.



(a) Grids instances.



(b) Mastermind instances.

Fig. 6: The run-time of *m-AOBB* in seconds as a function of number of solutions m for grid and mastermind instances.

8 Conclusion

The paper shows how Best First search algorithm for finding one optimal solution can be extended for finding the m-best solutions. We provide theoretical analysis of soundness and completeness and show that m-A^* is optimally efficient compared with any other algorithm that searches the same search space using the same heuristic function.

We then extend our analysis to Depth First Branch and Bound and apply the algorithms to graphical models optimization tasks, provide worst-case bounds on the search space explored by the m-best algorithms and characterize the added pruning power associated with the heuristic evaluation function. Preliminary empirical evaluation of *m-AOBB* shows that the algorithm often scales with the number of solutions m significantly better than the theory suggests.

We also present *BE-Greedy-m-BF*, a hybrid of variable-elimination and Best First Search scheme, that, interestingly, has the best time complexity amongst m-best algorithms known to us.

Acknowledgement

This work was supported by NSF grant IIS-1065618.

References

1. E. Charniak and S.E. Shimony. Cost-based abduction and map explanation. *Artificial Intelligence*, 66(2):345–374, 1994.
2. A. Darwiche, R. Dechter, A. Choi, V. Gogate, and L. Otten. Results from the probablistic inference evaluation of UAI08, a web-report in http://graphmod.ics.uci.edu/uai08/Evaluation/Report. *In: UAI applications workshop*, 2008.
3. R. Dechter and R. Mateescu. AND/OR search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
4. R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32:506–536, 1985.
5. David Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998.
6. N. Flerova, R. Dechter, and E. Rollon. Bucket and mini-bucket schemes for m best solutions over graphical models. In *GKR'11 Workshop*, 2011.
7. K. Kask and R. Dechter. A general scheme for automatic search heuristics from specification dependencies. *Artificial Intelligence*, pages 91–131, 2001.
8. E.L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401–405, 1972.
9. R. Marinescu and R. Dechter. AND/OR Branch-and-Bound search for combinatorial optimization in graphical models. *Artificial Intelligence*, 2009.
10. N. J. Nillson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, CA, 1980.
11. J. Pearl. *Heuristics: Intelligent Search Strategies.* Addison-Wesley, 1984.
12. T. Schiex. Arc consistency for soft constraints. *Principles and Practice of Constraint Programming (CP2000)*, pages 411–424, 2000.