Search Algorithms for m Best Solutions for Graphical Models

Rina Dechter and **Natalia Flerova**
University of California Irvine
USA

Radu Marinescu
IBM Research
Dublin, Ireland

Abstract

The paper focuses on finding the m best solutions to combinatorial optimization problems using Best-First or Branch-and-Bound search. Specifically, we present m-A*, extending the well-known A* to the m-best task, and prove that all its desirable properties, including soundness, completeness and optimal efficiency, are maintained. Since Best-First algorithms have memory problems, we also extend the memory-efficient Depth-First Branch-and-Bound to the m-best task. We extend both algorithms to optimization tasks over graphical models (e.g., Weighted CSP and MPE in Bayesian networks), provide complexity analysis and an empirical evaluation. Our experiments with 5 variants of Best-First and Branch-and-Bound confirm that Best-First is largely superior when memory is available, but Branch-and-Bound is more robust, while both styles of search benefit greatly when the heuristic evaluation function has increased accuracy.

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 combinatorial optimization tasks. In this paper, we explore the extension of such search algorithms to finding the m best solutions. We are interested in applying such algorithms to optimization tasks over graphical models, such as weighted CSPs and the most probable explanation (MPE) over probabilistic networks. These tasks arise in many applications, for example, procurement auction problems, biological sequence alignment or 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 sound and complete when guided by an admissible 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 (Dechter and Pearl 1985). Best-First Search and its most famous variant A* are also known to require significant memory and therefore substantial research went into trading memory and time, yielding schemes such as iterative deepening A* (Pearl and Korf 1987).

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

A 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 were developed for graphical models exploring the model's AND/OR search tree or the context-minimal AND/OR search graph (Dechter and Mateescu 2007), while using heuristics generated either by the mini-bucket scheme or through soft arc-consistency schemes (Marinescu and Dechter 2009a; 2009b; Schiex 2000).

The contributions of our paper are in extending both A* and Branch-and-Bound algorithms to the m best solutions, analyzing their performance analytically and empirically. Specifically, 1) we show (in Section 3) that m-A*, our proposed A* for the m-best task, inherits all A* desirable properties, most significantly it has an optimal performance; 2) we discuss an m-best Branch-and-Bound extension (in Section 4); 3) we discuss the extension of both algorithms to graphical models by exploring the AND/OR search spaces (in Section 5). 4) Empirical evaluations (in Section 7) confirm the superiority of m-A*, when memory is available, and show that otherwise, Branch-and-Bound provides a robust alternative for solving more problem instances. The dependency of the search efficiency on the heuristic strength is shown to be more pronounced as the number of solutions sought increases. Finally, 5) we show that a hybrid of bucket-elimination and search yields a scheme (called BE-Greedy-m-BF) that is superior to all other competing graphical models algorithms worst-case wise. This scheme suffers severely from memory issues over dense graphs, far more than A* schemes.

Two earlier works that are most relevant and provide the highest challenge to our work are by (Nilsson 1998) and (Aljazzar and Leue 2011) of which we learned only recently.

- Nilsson proposed a junction-tree based message passing scheme that iteratively finds the m best solutions and claimed that it has the best runtime complexity among m-best schemes for graphical models. Our analysis (not included here) can show that indeed Nilsson's scheme is very similar to our BE-Greedy-m-BF (Section 5.2). So, while we were unable to run Nilsson's code directly, we compare and contrast with BE-Greedy-m-BF. In particular this scheme is not feasible for problems having a large induced width, also known as treewidth.

- In the recent work (Aljazzar and Leue 2011) proposed K*, an A* search scheme for finding the k shortest paths that is interleaved with Breadth-First search. They use a specialized data structure and it is unclear if it can be extended straightforwardly to graphical models, a point we will leave to future work.

We will further elaborate and contrast our methods with related work towards the end of the paper. For lack of space all proofs are omitted.

2 Background

Consider 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 n_0 to a goal (Nillson 1980), where the cost of a solution path is the sum of the weights or the product of the weights on its arcs. In this paper, we explore extensions of Best-First Search (BFS) and Depth-First Branch-and-Bound (BnB) search to finding the m best solutions for general search spaces, and then apply those algorithms to search over graphical models.

BFS seems to be the most suitable algorithm to be extended to the m-best task. It explores the search space using a heuristic evaluation function $f(n)$ that estimates the best cost solution path that passes through each node n. It is known that when $f(n)$ is a lower bound on the optimal cost path the algorithm terminates with an optimal solution.

The most popular variant of BFS is A*, whose heuristic evaluation function is $f(n) = g(n) + h(n)$, where $g(n)$ 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 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 m Best Solutions

As was already noted, (e.g., (Charniak and Shimony 1994)), the extension of a BFS algorithm, including A*, to the task of finding the m best solutions seems simple: rather than terminating with the first solution found, the algorithm continues searching until it generates m solutions. In the following section we prove that indeed these solutions are the m best ones and that this simple scheme, that we call m-A*, is actually the best one among all its competitors under certain conditions.

3.1 Algorithm m-A*

Algorithm 1 provides a high level description of a tree-search variant which we call m-A*. The algorithm expands nodes in increasing value of f in the usual A* manner. For simplicity, we specify the algorithm under the assumption that h is consistent, but it can be extended to general admissible heuristics in the usual way. 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 at all wasteful when the heuristic function is consistent. In the subsequent sections we establish the following properties of m-A*, corresponding to the known properties of A* (Nillson 1980):

Algorithm 1: m-A*

```
Data: 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=$\emptyset$ and a tree $Tr = \emptyset$
Result: the $m$ best solutions
1 $i = 1$ ($i$ counts the current solution being searched for);
2 OPEN $\leftarrow \{n_0\}$; $f(n_0) = h(n_0)$; make $n_0$ the root of $Tr$;
3 If OPEN is empty then 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 Else expand $n$, generating all its children $Ch$. Compute
  $g(n') = g(n) + c(n, n')$; $f(n') = g(n') + h(n'), \forall n' \in Ch$ ;
7 If $n'$ already appears in OPEN or CLOSED $m$ times, discard
  node $n'$, else attach from each $n'$ in $Ch$ a pointer back to $n$ in
  $Tr$. Insert $n'$ into the right place in OPEN based on $f(n')$;
8 Go to step 3;
```

1. **Soundness and completeness**: m-A* terminates with the m best solutions generated in order of their costs.
2. **Optimal efficiency**: Any node that is surely expanded by m-A* must be expanded by any other sound and complete search algorithm.
3. **Optimal efficiency for consistent heuristics**: When the heuristic function is consistent m-A* expands each node at most m times.
4. **Dominance**: Given two heuristic functions h_1 and h_2, s.t. $\forall n$ $h_1(n) < h_2(n)$, m-A$^*_1$ will expand every node surely expanded by m-A$^*_2$, where m-A$^*_i$ is using heuristic h_i.

3.2 m-A* is Sound and Complete

We know that if provided with an admissible heuristic, A* will surely expand 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. We denote by C^*_i the i^{th} best solution cost and 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 $SS_1 \subset SS_2 \ldots \subset SS_i \ldots \subset SS_m$ and $C^*_1 \leq C^*_2 \ldots \leq C^*_m$.

Because at any point before the algorithm generates the i-best solution there is always a node in OPEN along each of the j-best solution path for $j = i, ..., m$ and because such a node must satisfy that $f(n) \leq C^*_i$, it follows that:

Proposition 1 *At any time before m-A* expands a goal node on the i^{th} best solution path, there is a node n' in OPEN satisfying $f(n') \leq C^*_i$.*

From this it follows that:

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

3.3 m-A* is Optimally Efficient

Algorithm A* is known to be optimally efficient (Dechter and Pearl 1985). 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*, namely nodes satisfying that they end a path from the root π s.t. $\forall n' \in \pi$, $f(n') < C^*$. 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 every 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 such that $f(n') < C^*_m$ $\forall n' \in \pi_{0..n}$.*

Similarly to (Dechter and Pearl 1985) 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, can miss one of the m best solutions when applied to a slightly different problem, and therefore, contradicts completeness.

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.*

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:*
1. *The maximum number of copies of the same node in OPEN or CLOSED can be bounded by m.*
2. *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 Search Space

The sizes of search space explored by m-A* for different levels of m are obviously monotonically increasing with m.

Proposition 2 *Given a search graph:*
1. *Any node expanded by i-A* is expanded by j-A* if $i < j$, if both use the same tie-breaking rule.*
2. *The set $S(i,j) = \{n|C^*_i < f(n) < C^*_j\}$ will surely be expanded by j-A* and surely not be expanded by i-A*.*
3. *If $C^*_i = C^*_j$, the number of nodes expanded by i-A* and j-A* 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.

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 found. 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 only on j-optimal paths $1 \leq j \leq m$.*

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 an 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, resulting in a number of nodes expansions bounded by $m \cdot n$, when n bounds the solution length. We can express the number of expanded nodes $\#N$ as $\#N = \sum_i \#N_i$ nodes, where $\#N_i$ is the length of the i-optimal solution path. We get therefore,

Theorem 5 *When m-A* has access to $h = h^*$, then, using a tie-breaking rule in favor of deeper nodes, it expands at most $\#N \leq m \cdot n$, when n is the maximum solution length.*

4 Branch-and-Bound for m Best Solutions

Branch-and-Bound explores the search space in a Depth-First manner. The algorithm maintains the best cost solution encountered so far, U, which is an *upper bound* on the exact minimum cost. Like A*, B&B uses a heuristic function $h(n)$ which underestimates the best cost solution below n, yielding a lower bound evaluation function $f(n)$. Therefore, whenever $f(n) > U$, search below n is pruned.

The main difference between B&B for one solution and its m-best extension m-BB is in the pruning condition. Let $U_1 \leq U_2 \leq ... \leq U_m$ denote the costs of the m best solutions encountered thus far, which m-BB maintains in a sorted list. Algorithm m-BB prunes a subproblem below node n iff $f(n) \geq U_m$.

When m-BB terminates, it outputs the m best solutions to the problem. We can show that when the search space explored is a graph and m-BB caches solutions, it has a runtime overhead of $O(m \cdot \log m)$ for each cachable node in the search space compared with both 1-BB that uses caching and m-BB without caching, respectively. The main reason is that we may have to maintain a sorted list of up to m solutions below each cachable node.

5 Application to Graphical Models

A *graphical model* is a tuple $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \sum)$, where $\mathbf{F}$ is a set of real-valued local cost functions over subsets of discrete variables $\mathbf{X}$, called scopes, with finite domains $\mathbf{D}$. The common optimization task is to find $\min_{\mathbf{X}} \sum_i f_i$ or $\max_{\mathbf{X}} \prod_i f_i$ (aka MPE). The scopes of $\mathbf{F}$ imply a *primal graph* G with certain *induced width* and a pseudo tree $\mathcal{T}$ of G that guides an AND/OR search space (Dechter and Mateescu 2007). An **AND/OR search tree** $\mathcal{S}_{\mathcal{T}}$ associated with $\mathcal{T}$ respects its structure and consists of 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 **F**. A complete assignment to the problem variables **X** is represented by a *solution tree* S of $\mathcal{S}_{\mathcal{T}}$ and the cumulative weight of S's edges corresponds to the cost of the assignment. We are interested in finding both the m best solution costs and their assignments.

The size of the AND/OR search tree based on $\mathcal{T}$ is $O(Nk^h)$, where N is the number of variables, k bounds the domain sizes, h is the height of $\mathcal{T}$. Identical subproblems can be identified and merged to obtain the **context minimal AND/OR search graph** which can be searched using additional memory and has size $O(Nk^{w^*})$, w^* being the induced width of G along a depth first traversal of $\mathcal{T}$. State of the art algorithms for solving optimization problems over graphical models are AOBB (AND/OR Branch-and-Bound) and AOBF (AND/OR Best-First) (Marinescu and Dechter 2009a; 2009b), that use the mini-bucket heuristic known to be admissible and consistent (Dechter and Rish 2003).

5.1 m-BB and m-A* for Graphical Models

Algorithm m-BB exploring the AND/OR search space is called m-AOBB. At any AND node n the m best solutions to the subproblems rooted at its children need to be combined and the best m out of the combined results should be chosen, introducing a time overhead of $O(deg \cdot m \log m)$ per AND node, where deg is the degree of n in the pseudo tree. In the absence of caching (i.e. exploring AND/OR tree) only up to m partial solutions need to be stored in memory, enabling m-AOBB to operate in linear space. Caching introduces both space and, as noted in Section 4, time overhead of $O(m \cdot \log m)$ per cachable (OR) node. For a problem with N variables:

Theorem 6 *The tree version of m-AOBB has time complexity of $O(Nk^h deg \cdot m \log m)$ and space complexity of $O(nm)$, where h is the height of the pseudo-tree. The graph version of m-AOBB has time and space complexity of $O(Nk^{w^*} deg \cdot m \log m)$, where w^* is the induced width of the ordering that guides the search.*

For the m-AOBF the overhead comes from expanding and storing up to m copies of each node. Since a node is only copied when a new path to it is discovered, this overhead pretains only to the graph-based version of the algorithm.

Theorem 7 *The time and space complexity of tree version of m-AOBF is bounded by $O(Nk^h)$. The graph version of m-AOBF has time and space complexity of $O(Nmk^{w*})$.*

5.2 Algorithm BE-Greedy-m-BF

Since an exact heuristic for graphical models can be generated by the Bucket Elimination (BE) algorithm (Kask and Dechter 2001), we can use the idea suggested in Section 3.6, yielding BE-Greedy-m-BF. The algorithm first generates the exact heuristics using BE and then applies m-A* (or m-AOBF) using these exact heuristics.

Theorem 8 *Let $\mathcal{M} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \sum)$ be a graphical model. The complexity of BE-Greedy-m-BF is $O(Nk^{w*} + mN)$.*

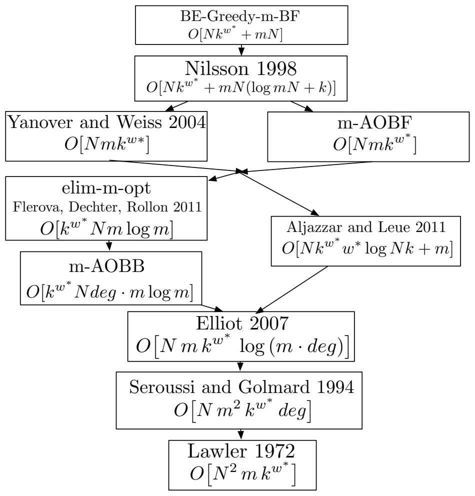
BE-Greedy-m-BF
$O[Nk^{w^*} + mN]$
Nilsson 1998
$O[Nk^{w^*} + mN(\log mN + k)]$
Yanover and Weiss 2004
$O[Nmk^{w*}]$
m-AOBF
$O[Nmk^{w^*}]$
elim-m-opt
Flerova, Dechter, Rollon 2011
$O[k^{w^*} Nm \log m]$
Aljazzar and Leue 2011
$O[Nk^{w^*} w^* \log Nk + m]$
m-AOBB
$O[k^{w^*} Ndeg \cdot m \log m]$
Elliot 2007
$O\left[N\, m\, k^{w^*} \log\,(m \cdot deg)\right]$
Seroussi and Golmard 1994
$O\left[N\, m^2\, k^{w^*}\, deg\right]$
Lawler 1972
$O\left[N^2\, m\, k^{w^*}\right]$

Figure 1: Complexity comparison. A parent node in the graph has a better complexity than its children.

It follows that the worst case complexity of the resulting hybrid scheme is superior to any other known m-best algorithm for graphical models.

6 Related Work

It is possible to distinguish three main approaches employed by earlier m-best exact algorithms. First one assumes finding solutions iteratively, an idea made popular by (Lawler 1972), who provided a general iterative scheme for extending any given optimization algorithm to the m-best task, and more recently improved by (Yanover and Weiss 2004) and (Nilsson 1998). Another approach lies in using dynamic programming or variable elimination to directly obtain the m best solutions (Seroussi and Golmard 1994; Elliott 2007; Flerova, Dechter, and Rollon 2011). Third idea is to use search to solve a related task of finding k shortest paths. The majority of schemes in the latter category can not be applied to the m-best combinatorial optimization task, since they assume the availability of an explicit search graph, which for most practical problems is too large. However, this problem was overcome in the recently work by (Aljazzar and Leue 2011).

Figure 1 compares the worst-case time complexity between our schemes and earlier m-best work over graphical models. A parent node in the graph has a better complexity than its children. In many cases the complexity analysis is ours. Note that worst-case analysis cannot capture the optimality of m-A* that we proved above.

7 Experiments

We evaluate the performance of Best-First and Depth-First search algorithms on finding the m best solutions to a weighted CSP and on the m most probable explanations in Bayesian networks. The benchmark problems considered include problem instances derived from genetic linkage analysis networks, n-by-n grid networks used during the UAI 2008 competition and ISCAS'89 digital circuits, all online at *http://graphmod.ics.uci.edu/*. The algorithms were

instance	algorithm	$i = 10$						$i = 16$						$i = 22$					
(n, k, w^*, h)		$m = 1$		$m = 10$		$m = 100$		$m = 1$		$m = 10$		$m = 100$		$m = 1$		$m = 10$		$m = 100$	
		time	nodes	time	nodes	time	nodes	**genetic linkage networks (Bayesian networks)**						time	nodes	time	nodes	time	nodes
	m-AOBB	-		-		-		-		-		-		-		-		-	
pedigree7	m-AOBF	out		out		out		out		out		out		out		out		out	
(1069, 5)	m-BB	-		-		-		-		-		-		**25752**	1.9B	**25567**	1.9B	**25548**	1.9B
(47, 204)	m-BF	out		out		out		out		out		out		out		out		out	
	m-AOBB	938.42	4.9M	5133.8	11.2M	-		13.75	50.3K	87.35	139.1K	3196.81	327.9K	10.60	310	11.74	3.1K	77.50	11.0K
pedigree23	m-AOBF	out		out		out		27.66	328.8K	27.73	329.7K	out		7.23	719	7.37	1.4K	8.88	8.0K
(403, 5)	m-BB	492.13	65.5M	497.04	66.6M	549.36	73.6M	7.52	832.7K	8.88	953.1K	19.80	1.6M	**7.01**	785	7.27	8.6K	9.29	110.1K
(21, 64)	m-BF	120.63	7.5M	120.55	7.5M	146.65	8.9M	**1.54**	54.0K	**1.61**	55.8K	**2.35**	85.3K	7.08	630	**7.17**	2.3K	7.76	19.3K
	BE+m-BF	**7.11**	630	**7.24**	2.5K	**7.68**	19.3M	7.11	630	7.24	2.5K	7.68	19.3K	7.11	630	7.24	2.3K	**7.68**	19.3K
	m-AOBB	35.56	164.4K	137.23	292.9K	2519.8	607.7K	366.67	6.5K	371.53	13.7K	513.47	29.0K	out		out		out	
pedigree37	m-AOBF	out		out		out		out		out		out		out		out		out	
(798, 5)	m-BB	10.87	854.4K	15.54	947.9K	87.89	2.4M	38.08	14.7K	38.23	40.4K	66.24	607.5K	out		out		out	
(20, 72)	m-BF	**5.56**	202.2K	**5.93**	205.6K	**8.43**	242.3K	**37.12**	5.6K	**37.56**	9.0K	**39.91**	44.4K	out		out		out	
	m-AOBB	**2060.2**	15.5M	**3916.8**	17.8M	**23281**	23.2M	out		out		out		out		out		out	
pedigree38	m-AOBF	out		out		out		out		out		out		out		out		out	
(725, 5)	m-BB	-		-		-		out		out		out		out		out		out	
(16, 52)	m-BF	out		out		out		out		out		out		out		out		out	
	m-AOBB	2107.8	9.9M	4148.0	11.9M	38089	14.7M	27.16	117.2K	80.69	184.4K	977.78	253.5K	**11.69**	958	**12.72**	2.9K	84.23	8.9K
pedigree39	m-AOBF	out		out		out		out		out		out		13.78	1.4K	14.62	2.4K	21.91	12.4K
(1273, 5)	m-BB	6874.6	592.3M	6833.8	592.4M	6856.1	592.8M	17.58	1.5M	18.93	1.5M	76.59	2.6M	13.12	2.4K	13.17	5.4K	15.86	56.8K
(20, 78)	m-BF	out		out		out		**10.17**	474.8K	**10.29**	476.4K	**11.33**	491.2K	13.22	1.7K	13.35	3.3K	14.51	18.0K
	BE+m-BF	**13.29**	1.7K	**13.46**	3.3K	**14.42**	18.1K	13.29	1.7K	13.46	3.2K	14.42	18.0K	13.29	1.7K	13.46	3.3K	**14.42**	18.0K
grid networks (Bayesian networks)																			
	m-AOBB	-		-		-		1181.8	5.2M	2157.8	6.9M	-		19.63	789	21.50	3.3K	111.61	19.6K
g-50-18-5	m-AOBF	out		out		out		out		out		out		13.65	5.8K	13.97	8.8K	15.75	24.6K
(324, 2)	m-BB	-		-		-		1135.7	171M	1316.9	199M	1840.1	277M	**13.52**	16.5K	**13.56**	27.2K	14.71	114.7K
(24, 84)	m-BF	out		out		out		out		out		out		13.65	8.2K	13.68	11K	**14.01**	26.3K
	BE+m-BF	**37.98**	324	**38.00**	1.7K	**38.46**	12.1K	**37.98**	324	**38.00**	1.7K	**38.00**	12.1K	37.98	324	38.00	1.7K	38.46	12.1K
	m-AOBB	-		-		-		-		-		-		226.38	770.9K	654.08	1.5M	5126.0	3.5M
g-50-20-5	m-AOBF	out		out		out		out		out		out		out		out		out	
(400, 2)	m-BB	-		-		-		-		-		-		424.57	64.1M	512.80	78M	723.66	109.6M
(27, 93)	m-BF	out		out		out		out		out		out		**55.39**	2.9M	**63.69**	3.5M	**82.62**	4.9M
	m-AOBB	1076.5	4.8M	3070.7	8.9M	-		7.63	33K	59.64	135.4K	510.92	353.7K	22.02	722	24.98	5.8K	89.85	28.4K
g-75-18-5	m-AOBF	out		out		out		26.29	368.8K	out		out		13.33	1.3K	13.92	7.1K	17.26	33.2K
(324, 2)	m-BB	-		-		-		7.86	1.3M	15.62	2.5M	32.67	4.9M	**13.13**	1.5K	13.35	21K	15.95	196.3K
(24, 85)	m-BF	out		out		out		**3.57**	280.4K	**7.98**	639K	**17.95**	1.4M	13.27	465	**13.28**	2.9K	**13.87**	22.2K
	BE+m-BF	**38.05**	324	**38.05**	2.5K	**38.69**	21.4K	38.05	324	38.05	2.5K	38.69	21.4K	38.05	324	38.05	2.5K	38.69	21.4K
	m-AOBB	-		-		-		1635.1	6.2M	-		-		30.94	17.5K	52.70	50K	347.02	132.6K
g-75-19-5	m-AOBF	out		out		out		out		out		out		out		out		out	
(361, 2)	m-BB	-		-		-		-		-		-		**14.44**	36.7K	15.31	90.5K	20.81	467.8K
(25, 85)	m-BF	out		out		out		out		out		out		14.51	11.7K	**14.56**	16.6K	**15.09**	38.7K
	BE+m-BF	**143.11**	361	**143.11**	2.3K	**144.11**	16.9K	**143.11**	361	**143.11**	2.3K	**144.11**	16.9K	143.11	361	143.11	2.3K	144.11	16.9K
	m-AOBB	-		-		-		214.21	1M	345.74	1.2M	1514.7	1.5M	47.19	33K	59.96	51.1K	246.82	88K
g-90-20-5	m-AOBF	out		out		out		out		out		out		19.97	17.8K	21.66	32.1K	29.65	96.5K
(400, 2)	m-BB	-		-		-		52.96	6.4M	67.87	8.2M	108.22	12.9M	18.44	95.5K	19.34	158.4K	24.80	498.2K
(27, 99)	m-BF	out		out		out		**17.09**	1.1M	**25.00**	1.6M	**45.46**	2.9M	**18.17**	3.9K	**18.19**	7K	**18.56**	21.3K
ISCAS networks (Weighted CSPs)																			
		$i = 6$						$i = 12$						$i = 18$					
	m-AOBB	898.83	4.1M	4354.9	7.9M	-		0.49	2.4K	2.48	4.9K	90.57	13.5K	**0.52**	433	1.52	2.7K	40.17	8.6K
c432	m-AOBF	**1.72**	18.9K	**2.05**	19.9K	**30.46**	290.1K	0.07	288	0.25	1K	1.86	7.5K	0.69	287	0.85	1K	2.54	7.4K
(432, 2)	m-BB	-		-		-		3.08	153.5K	6.31	304K	41.52	2.1M	1.09	22.9K	2.85	107.6K	35.35	1.7M
(28, 46)	m-BF	279.60	12.2M	281.03	12.2M	out		**0.03**	507	**0.14**	3.7K	**1.22**	34.2K	0.64	432	**0.77**	3.6K	**1.84**	33.9K
	m-AOBB	-		-		-		449.59	1.1K	2900.2	2.4M	-		28.91	74.7K	60.57	107.4K	1853.4	347.6K
s953	m-AOBF	**11.91**	106K	**11.91**	106K	**23.35**	211.4K	0.29	2.4K	0.29	2,477	**0.46**	4K	2.92	600	2.93	648	3.06	1.3K
(441, 2)	m-BB	-		-		-		2013.5	147.2M	1976.1	147.5M	2010.4	151.7M	7.00	315.7K	7.05	317.5K	9.91	487.5K
(72, 99)	m-BF	out		out		out		**0.09**	1.5K	**0.10**	1.6K	0.47	16.6K	**2.91**	656	**2.87**	732	**2.97**	4.4K
	m-AOBB	-		-		-		289.72	835.5K	625.10	1.2M	6652.2	2.8M	26.53	66.6K	48.10	88K	681.16	216.8K
s1196	m-AOBF	out		out		out		5.62	81.9K	**13.66**	155K	out		3.70	2.2K	7.01	21.5K	out	
(562, 2)	m-BB	-		-		-		107.80	7.3M	110.24	7.4M	237.73	15.7M	8.02	338.4K	8.78	364.7K	20.16	877.4K
(54, 110)	m-BF	out		out		out		**0.82**	34.2K	14.84	539K	**49.03**	1.9M	**3.43**	3.9K	**3.74**	17.6K	**4.65**	46.1K
	m-AOBB	-		-		-		-		-		-		409.05	984K	1094.1	1.5M	-	
s1238	m-AOBF	out		out		out		out		out		out		3.67	3.1K	5.16	9.2K	25.60	96.9K
(541, 2)	m-BB	-		-		-		-		-		-		103.53	7.9M	106.62	8M	165.55	10.9M
(58, 100)	m-BF	out		out		out		**76.61**	2.8M	**75.68**	2.8M	**351.05**	12.9M	**3.60**	7.1K	**3.62**	9.8K	**4.92**	51.4K
	m-AOBB	5095.3	14.7M	-		-		0.81	2.8K	49.44	56.3K	4118.8	394.2K	**3.82**	1.4K	25.14	23.9K	1956.7	175.9K
s1488	m-AOBF	**12.38**	42.8K	out		out		0.38	1.5K	**2.19**	9.7K	**17.91**	70.8K	5.02	4.2K	8.09	20K	out	
(667, 2)	m-BB	-		-		-		6.00	231.8K	130.39	4.2M	504.33	19.1M	6.82	86.1K	173.27	5M	702.75	21.4M
(46, 69)	m-BF	out		out		out		**0.14**	939	3.87	84.6K	33.62	692.7K	4.37	783	**4.81**	11.6K	**9.02**	110.9K

Table 1: CPU time (in seconds) and number of nodes expanded for the genetic linkage, grid and ISCAS networks. An '-' stands for exceeding the time limit (12h - pedigrees, 2h - grids and ISCAS networks). 'out' indicates out of 4GB memory. Algorithm BE+m-BF is shown in the corresponding table row only when it solved that problem instance, and is omitted otherwise.

implemented in C++ (32-bit) and the experiments were run on a 2.6GHz quad-core processor with 12GB of RAM.

We can distinguish 4 algorithms based on the particular order of node expansions and the structure of the underlying search space: two that explore a regular OR search tree, denoted by m-BF and m-BB, and two that explore an AND/OR search tree, namely m-AOBF and m-AOBB. So far we implemented algorithms that explore a search tree and not a graph. In the case of Best-First Search this is justified by theory. Caching for Depth-First Branch-and-Bound allows to expand less nodes, but could bring significant space overhead, and this will be studied in future work. We also ran the

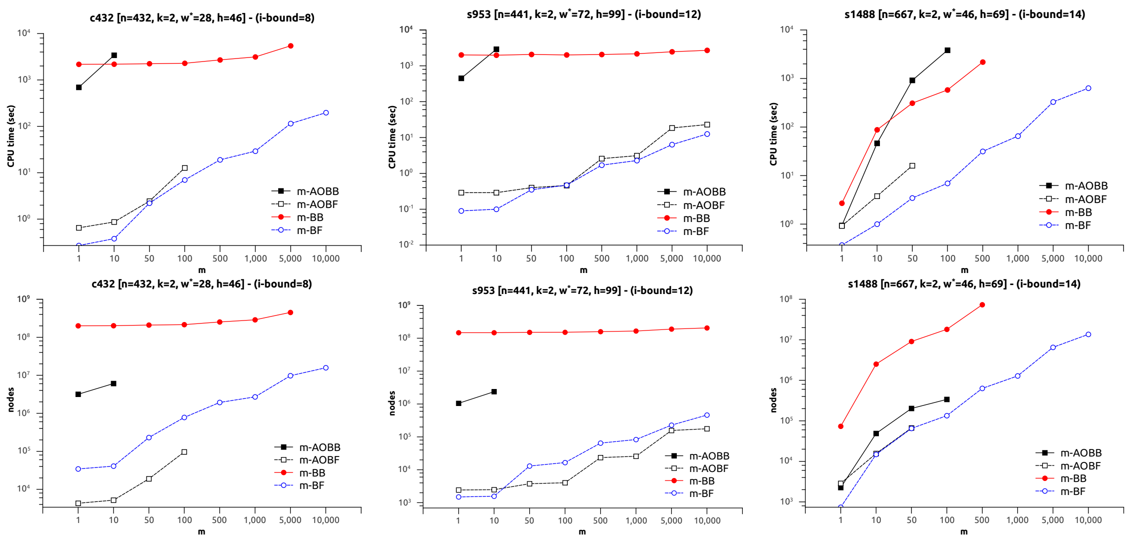

Figure 2: CPU time and nodes expanded as a function of m for the ISCAS networks c432, s953 and s1488, respectively.

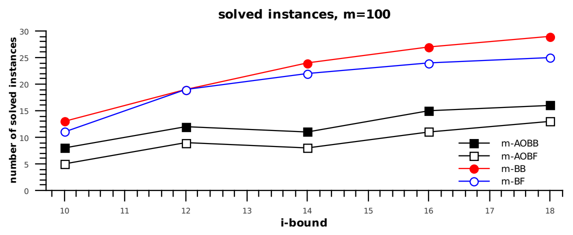

Figure 3: Number of solved instances as a function of the mini-bucket i-bound, $m = 100$.

hybrid BE-Greedy-m-BF described in Section 5.2 (denoted here as BE+m-BF). All algorithms were guided by pre-compiled mini-bucket heuristics (Kask and Dechter 2001; Marinescu and Dechter 2009a) and were restricted to a static variable ordering which was computed using a min-fill heuristic iteratively and stochastically (Kask et al. 2011).

We report the CPU time (in seconds) and the number of nodes expanded during search and all the relevant parameters (n, k, w^*, h). The best runtime performance points are highlighted.

Table 1 shows the results obtained for 5 hard linkage networks, 5 grid networks and 5 ISCAS circuits. The columns are indexed by the mini-buckets i-bound, the number of solutions considered and for each we report the time and the number of nodes expanded. The best time for each i-bound is shown in bold and underlined. We observed that:

1 *Best First vs. Branch and Bound.* Algorithm m-BF offers the overall best performance when sufficient memory is available, winning on 12 out of the 15 test instances shown. For example, on the s1196 circuit, m-BF with $i = 12$ found 100 solutions in 49 seconds, about 5 and 133 times faster than m-BB and m-AOBB, respectively. Algorithm m-BB is otherwise more robust and, therefore, is able to find all 100 solutions to the hardest instance in the dataset (eg, pedigree7) in about 7 hours, while others aborted due to memory or time limit. Figure 3 shows that overall m-BB solves more instances for various i-bounds.

2 *Impact of the AND/OR search space:* the AND/OR search algorithm m-AOBB is inferior to m-BB time-wise on most instances even though it expands significantly less nodes (eg, c432). Its powerful decomposition strategy does not translate into time savings, apparently due to overhead of combining m partial solutions from different child nodes. Figure 4 shows the time spent per node, averaged over the pedigree problems. Still, for some instances m-AOBB is more efficient than m-BB, for example, pedigree38. As for Best-First search, m-AOBF is competitive and sometimes significantly faster than m-BF when enough memory is available (eg, c432, s953), but it runs out of memory quicker than m-BF because of additional data structures needed to maintain the AND/OR tree and its OPEN list.

3 *Impact of the heuristic information:* the B&B algorithms typically outperform the BFS algorithms for less accurate heuristics (smaller i-bounds), where the latter run out of memory relatively quickly (eg, pedigree38, pedigree39). As the heuristic accuracy increases (larger i-bounds), BFS outperforms dramatically B&B search (eg, s953, s1196). However, for the most accurate heuristics (largest i-bounds), the difference between B&B and BFS is often smaller, perhaps, because B&B finds almost optimal solutions fast and therefore, like Best-First search, will not explore solutions whose evaluation function is above the optimal one.

4 *Impact of the number of solutions:* algorithms m-BF and m-BB are able to scale to much larger m values than 100 due to their bounded computational overhead compared with m-AOBB/m-AOBF. Figure 2 displays the CPU time and number of nodes expanded as a function of m, for three ISCAS circuits. Algorithm m-BF is the fastest algorithm across all reported m values, outperforming m-BB by up to four orders of magnitude while exploring a significantly smaller search space. Algorithm m-AOBF is competitive with m-BF but

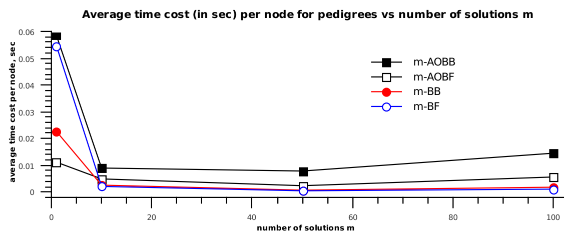

Figure 4: Average time (in sec) per node as a function of m.

only for relatively small m values, because of computational overhead issues. Algorithm m-AOBB scaled up to $m = 10$ on c432 and s953, and up to $m = 100$ on s1488, while exceeding the time limit for larger m values, even though the search space explored was far smaller than that of m-BB again, due to node overhead.

5 *Algorithm BE+m-BF:* solved only 5 test instances and failed on the rest due to the memory implied by the very large induced widths. We expect a similar performance from the variable elimination based approaches (e.g., (Nilsson 1998)), because they all require memory exponential in the induced width w^* and lack an efficient pruning mechanism.

8 Conclusion

Most of the work on finding m best solutions over graphical models was focused on either iterative schemes based on Lawler's idea, or on dynamic programming (e.g., variable-elimination or tree-clustering). In this paper we showed for the first time that for combinatorial optimization defined over graphical models the traditional heuristic search paradigms are superior. In particular, we extended Best-First and Branch-and-Bound search algorithms to solving the m-best optimization tasks, presenting m-A* and m-BB, and proved analytically that m-A* is superior to any other search scheme. We also introduced BE-Greedy-m-BF, a hybrid of variable elimination and Best-First scheme and showed that it has the best worst-case time complexity amongst all m-best algorithms over graphical models known to us. However, we empirically demonstrated the superiority of m-A* over the BE-Greedy-m-BF scheme in practice, and thus can infer general superiority over many of the other algorithms.

Finally, we also demonstrated empirically that since both Best-First Search and the elimination-based schemes (e.g., BE-Greedy-m-BF and *elim-m-opt* reported in (Flerova, Dechter, and Rollon 2011)) require too much memory when the graph is dense (e.g., the induced width w^* is high), a Branch-and-Bound scheme like m-BB, which can trade space for time, is a better option. Note, that our implementations of the search algorithms here were restricted to search spaces that are trees. One of our main future tasks is to explore extensions to search spaces that are graphs.

Acknowledgement

This work was supported by NSF grant IIS-1065618.

References

Aljazzar, H., and Leue, S. 2011. K : A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence* 175(18):2129–2154.

Charniak, E., and Shimony, S. 1994. Cost-based abduction and map explanation. *Artificial Intelligence* 66(2):345–374.

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

Dechter, R., and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM* 32:506–536.

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

Elliott, P. 2007. Extracting the K Best Solutions from a Valued And-Or Acyclic Graph. Master's thesis, Massachusetts Institute of Technology.

Flerova, N.; Dechter, R.; and Rollon, E. 2011. Bucket and mini-bucket schemes for m best solutions over graphical models. In *Graph structures for knowledge representation and reasoning workshop*.

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

Kask, K.; Gefland, A.; Otten, L.; and Dechter, R. 2011. Pushing the power of stochastic greedy ordering schemes for inference in graphical models. In *25th Conference on Artificial Intelligence (AAAI)*, 54–60.

Lawler, E. 1972. 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.

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

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

Nillson, N. J. 1980. *Principles of Artificial Intelligence*. Tioga, Palo Alto, CA.

Nilsson, D. 1998. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing* 8(2):159–173.

Pearl, J., and Korf, R. 1987. Search techniques. *Annual Reviews of Computer Science* 2:451–467.

Schiex, T. 2000. Arc consistency for soft constraints. *International Conference on Principles and Practice of Constraint Programming (CP)* 411–424.

Seroussi, B., and Golmard, J. 1994. An algorithm directly finding the K most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning* 11(3):205–233.

Yanover, C., and Weiss, Y. 2004. Finding the M Most Probable Configurations Using Loopy Belief Propagation. In *Advances in Neural Information Processing Systems 16*. The MIT Press.