Bucket and Mini-Bucket Schemes for M Best Solutions over Graphical Models

Natalia Flerova[1], Emma Rollon[2] and Rina Dechter[1]

[1] University of California Irvine
[2] Universitat Politecnica de Catalunya

Abstract. The paper focuses on the task of generating the first m best solutions for a combinatorial optimization problem defined over a graphical model (e.g., the m most probable explanations for a Bayesian network). We show that the m-best task can be expressed within the unifying framework of semirings making known inference algorithms defined and their correctness and completeness for the m-best task immediately implied. We subsequently describe *elim-m-opt*, a new bucket elimination algorithm for solving the m-best task, provide algorithms for its defining combination and marginalization operators and analyze its worst-case performance. An extension of the algorithm to the mini-bucket framework provides bounds for each of the m-best solutions. Empirical demonstrations of the algorithms with emphasis on their potential for approximations are provided.

1 Introduction

The aim in combinatorial optimization is to find an optimal solution, minimum or maximum, of an objective function. In many applications, however, it is desirable to obtain not just a single optimal solution, but a set of the first m best solutions for some integer m. Such a set can be useful, for example, in assessing the sensitivity of the optimal solution to variation of the problem's parameters, a task that arises, for instance, in biological sequence alignment. Sometimes a set of diverse assignments with approximately the same cost is needed, as in reliable communication network design and genetic linkage analysis (haplotyping). In some areas, such as in procurement auction problems, certain constraints of the problem might be too vague or too complicated to formalize. In such cases it may be more practical to first find several good solutions to a relaxed problem and then pick the one that satisfies all the additional constraints.

One of the earliest and most influential work on finding the m best solutions was developed by Lawler [18]. He provided a general scheme that extends any optimization algorithm to the m-best task. The idea is to compute the next best solution successively by finding a single optimal solution for a slightly different reformulation of the original problem that excludes the solutions generated so far. This approach has been extended and improved over the years and is still one of the primary strategies for finding the m best solutions. Other approaches are more direct, trying to avoid the repeated computation inherent to Lawler's scheme.

Our goal is to develop a direct extension of general optimization schemes, such as dynamic programming or search, to the m-best task. Our focus is on high dimensional discrete spaces and on objective functions that can be factorized using a collection of low dimensional functions. These formulations normally appear in the context of graphical models such as Bayesian networks, Markov networks and constraint networks.

Graphical models [20] are widely used knowledge representation schemes that give rise to probabilistic and deterministic combinatorial optimization tasks, such as finding the most likely configuration of a Bayesian network (called MAP or MPE) or finding a solution that violates the least number of constraints in the CSP (called max-CSP). In these models knowledge is represented by a collection of local functions, each defined on a small subset of variables, whose interaction can be captured by a graph. The task is to optimize a global objective function expressed as a combination (sum or product) of the local functions. Various graph-exploiting algorithms for solving these optimization tasks were developed in the past few decades. Such algorithms are often characterized as being either of *inference* type (e.g., message-passing schemes, variable elimination) or of *search* type (e.g., AND/OR search or recursive-conditioning). We limit our treatment here to the class of inference schemes as represented by the Bucket Elimination algorithm (BE) [8].

We will show that Bucket Elimination can be extended to compute the m best solutions by representing functions as vector functions and modifying of the algorithm's underlying combination and marginalization operators to vector functions. The extension is accomplished by formalizing the m-best task within the framework of semirings [22, 1, 17, 3]. This unifying formulation ensures the soundness and completeness of any algorithm applied to any problem that fits into the framework and in particular it implies that *elim-m-opt* solves the m-best optimization task.

Since the bucket elimination scheme can be approximated by the relaxation and bounding scheme of mini-bucket elimination (MBE) [11], we can extend *elim-m-opt* straightforwardly to a mini-bucket scheme *mbe-m-opt*, which computes a bound on each of the m best solutions. More significantly, we also show that the m bounds computed by *mbe-m-opt* can be used to tighten the bound on the first best solution since both are generated from the same relaxed problem. In particular, it can facilitate a scheme for tightening any heuristic generation scheme.

After providing some background in Section 2, we formulate the m-best task in the context of semirings in Section 3. Sections 4 and 5 describe algorithms *elim-m-opt* and *mbe-m-opt* respectively. We show the results of the empirical evaluation in Section 6 and compare our schemes with related work in Section 7. Section 8 provides a summary and discusses future work.

2 Background

We consider problems expressed as graphical models. The graphical model framework provides a common formalism to model a broad spectrum of problems, and a collection of general algorithms to efficiently solve them. Examples of graphical models are Markov and Bayesian networks [20], constraint networks and influence diagrams. We next follow definitions as in [16].

Let $\mathbf{X} = (X_1, \ldots, X_n)$ be an ordered set of variables and $\mathbf{D} = (\mathbf{D}_1, \ldots, \mathbf{D}_n)$ an ordered set of domains. Domain $\mathbf{D}_i$ is a finite set of potential values for X_i. The assignment of variable X_i with $a \in \mathbf{D}_i$ is noted $(X_i = a)$. A tuple is an ordered set of assignments to different variables $(X_{i_1} = a_{i_1}, \ldots, X_{i_k} = a_{i_k})$. A *complete assignment* to all the variables in $\mathbf{X}$ is called a *solution*. Let t and s be two tuples having the same instantiations to the common variables. Their join, noted $t \cdot s$, is a new tuple which contains the assignments of both t and s (this notation is also used for multiplication, so we assume the meaning will be clear from the context). If t is a tuple over a set $\mathbf{T} \subseteq \mathbf{X}$ and $\mathbf{S}$ is a set of variables, then $t_{[\mathbf{S}]}$ is a relational projection of t on $\mathbf{S}$.

2.1 Valuation Structure and Functions

Let $\mathbf{A}$ be a set whose elements are called *valuations* (e.g., the naturals, the reals or the booleans). The set of valuations $\mathbf{A}$ admits two binary operations: $\otimes : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A}$ called *combination* and $\oplus : \mathbf{A} \times \mathbf{A} \rightarrow \mathbf{A}$ called *addition*. Both operators are associative and commutative. Typical combination operators are sum and product over numbers, and logical AND (i.e., $\wedge$) over booleans. Typical addition operators are min, max and sum over numbers and logical OR (i.e., $\vee$) over booleans.

The properties of a valuation structure can be described by means of an algebraic structure. In this paper we focus on an algebraic structure called *semiring*.

Definition 1 (semiring). *A* commutative semiring *is a triplet* $(\boldsymbol{A}, \otimes, \oplus)$ *such that:*

– $\otimes$ *and* $\oplus$ *are associative and commutative*
– $\otimes$ *distributes over* $\oplus$*, that is,* $(a \otimes b) \oplus (a \otimes c) = a \otimes (b \oplus c)$

A commutative semiring defines an ordering among its elements as follows. For any $a, b \in \mathbf{A}$, $a > b$ holds (i.e., a is *better* than b), if there exists $c \in \mathbf{A}$ such that $a = b \oplus c$.

We denote by $\mathbf{D}_\mathbf{Y}$ the set of tuples over a subset of variables $\mathbf{Y}$, also called the *domain* of $\mathbf{Y}$. Functions are defined on subsets of variables of $\mathbf{X}$, called scopes, and their range is a set of valuations $\mathbf{A}$. If $f : \mathbf{D}_\mathbf{Y} \rightarrow \mathbf{A}$ is a function, the scope of f, denoted $var(f)$, is $\mathbf{Y}$. In the following, we will use $\mathbf{D}_f$ as a shorthand for $\mathbf{D}_{var(f)}$.

Definition 2 (combination operator). *Let* $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}$ *and* $g : \boldsymbol{D}_g \rightarrow \boldsymbol{A}$ *be two functions. Their* combination, *noted* $f \bigotimes g$ *is a new function with scope* $var(f) \cup var(g)$*, s.t.*

$$\forall t \in \boldsymbol{D}_{var(f) \cup var(g)},\ (f \bigotimes g)(t) = f(t) \otimes g(t)$$

Definition 3 (marginalization operator). *Let* $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}$ *be a function and* $\boldsymbol{W} \subseteq \boldsymbol{X}$ *be a set of variables. The* marginalization *of* f *over* $\boldsymbol{W}$*, noted* $\Downarrow_{\boldsymbol{W}} f$*, is a function whose scope is* $var(f) - \boldsymbol{W}$*, s.t.*

$$\forall t \in D_{var(f) - \boldsymbol{W}}, (\Downarrow_{\boldsymbol{W}} f)(t) = \oplus_{t' \in D_W}(t \cdot t')$$

Example 1. Consider three variables X_1, X_2 and X_3 with domains $\mathbf{D}_1 = \mathbf{D}_2 = \mathbf{D}_3 = \{1, 2, 3\}$. Let $f(X_1, X_2) = X_1 X_2$ and $g(X_2, X_3) = 2X_2 + X_3$ be two functions. If the combination operator is product (i.e., $\times$), then $(f \otimes g)(X_1, X_2, X_3) = (f \times g)(X_1, X_2, X_3) = X_1 X_2 \times (2X_2 + X_3)$. If the marginalization operator is *max*, then $(f \Downarrow_{X_1})(X_2) = \max\{f(X_1 = 1, X_2), f(X_1 = 2, X_2), f(X_1 = 3, X_2)\} = \max\{1X_2, 2X_2, 3X_2\} = 3X_2$.

2.2 Graphical Model

Definition 4 (graphical model). *A* graphical model *is a tuple $\mathcal{M} = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes)$, where: $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ is a set of variables; $\boldsymbol{D} = \{\boldsymbol{D}_1, \ldots, \boldsymbol{D}_n\}$ is the set of their finite domains of values; $\boldsymbol{A}$ is a set of valuations $(\boldsymbol{A}, \otimes, \oplus)$; $\boldsymbol{F} = \{f_1, \ldots, f_r\}$ is a set of discrete functions, where $var(f_j) \subseteq \boldsymbol{X}$ and $f_j : \boldsymbol{D}_{f_j} \rightarrow \boldsymbol{A}$; and $\bigotimes$ is the combination operator over functions (see Definition 2). The graphical model $\mathcal{M}$ represents the function $C(\boldsymbol{X}) = \bigotimes_{f \in \boldsymbol{F}} f$.*

Definition 5 (reasoning task). *A* reasoning task *is a tuple $P = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$, where $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes)$ is a graphical model and $\Downarrow$ is a marginalization operator (see Definition 3). The reasoning task is to compute $\Downarrow_{\boldsymbol{X}} C(\boldsymbol{X})$.*

For a reasoning task $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$ the choice of $(\mathbf{A}, \otimes, \oplus)$ determines the combination $\bigotimes$ and marginalization $\Downarrow$ operators over functions, and thus the nature of the graphical model and its reasoning task. For example, if $\mathbf{A}$ is the set of non-negative reals and $\bigotimes$ is product, the graphical model is a Markov network or a Bayesian network. If $\Downarrow$ is $\max$, the task is to compute the Most Probable Explanation (MPE), while if $\Downarrow$ is sum, the task is to compute the Probability of the Evidence.

The correctness of the algorithmic techniques for computing a given reasoning task relies on the properties of its valuation structure. In this paper we consider reasoning tasks $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$ such that their valuation structure $(\mathbf{A}, \otimes, \oplus)$ is a semiring. Several works [22, 1, 17] showed that the correctness of inference algorithms over a reasoning task P is ensured whenever P is defined over a semiring.

Example 2. MPE task is defined over semiring $\mathbf{K} = (\mathtt{R}, \times, \max)$, a CSP is defined over semiring $\mathbf{K} = (\{0, 1\}, \wedge, \vee)$, and a Weighted CSP is defined over semiring $\mathbf{K} = (\mathtt{N} \cup \{\infty\}, +, \min)$. The task of computing the Probability of the Evidence is defined over semiring $\mathbf{K} = (\mathtt{R}, \times, +)$.

2.3 Bucket and Mini-Bucket Elimination

Bucket Elimination (BE) [8] (see Algorithm 1) is a well-known inference algorithm that generalizes dynamic programming for many reasoning tasks. The input of BE is a reasoning task $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$ and an ordering $o = (X_1, X_2, \ldots, X_n)$, dictating an elimination order for BE, from last to first. Each function from $\mathbf{F}$ is placed in the bucket of its latest variable in o. The algorithm processes the buckets from X_n to X_1, computing for each bucket X_i, noted $\mathbf{B}_i$, $\Downarrow_{X_i} \bigotimes_{j=1}^{n} \lambda_j$, where λ_j's are the functions in the $\mathbf{B}_i$, some of which are original f_i's and some are earlier computed messages. The result of the computation is a new function, also called *message*, that is placed in the bucket of its latest variable in the ordering o.

Depending on the particular instantiation of the combination and marginalization operators BE solves a particular reasoning task. For example, algorithm *elim-opt*, which solves the optimization task, is obtained by substitution of the operators $\Downarrow_X f = \max_{S-X} f$ and $\bigotimes_j = \prod_j$.

The message passing between buckets follows a bucket-tree structure.

Algorithm 1 Bucket elimination

```
Input: A reasoning task $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$; An ordering of variables $o = \{X_1, \dots, X_n\}$;
Output: A zero-arity function $\lambda_1 : \emptyset \rightarrow \mathbf{A}$ containing the solution of the reasoning task.
1: Initialize: Generate an ordered partition of functions in buckets $\mathbf{B}_1, \dots, \mathbf{B}_n$, where $\mathbf{B}_i$ contains all the functions whose highest variable in their scope is $X_i$.
2: Backward:
3: for $i \leftarrow n$ down to 1 do
4:     Generate $\lambda_i = (\bigotimes_{f \in \mathbf{B}_i} f) \Downarrow_{X_i}$
5:     Place $\lambda_i$ in the bucket $B_j$ where $j$ is the largest-index variable in $var(\lambda_i)$
6: end for
7: Return: $\lambda_1$
```

Definition 6 (bucket tree). *Bucket Elimination defines a* bucket tree, *where a node of the bucket is associated with its bucket variable and the bucket of each X_i is linked to the destination bucket of its message (called the parent bucket).*

The complexity of Bucket Elimination can be bounded using the graph parameter of *induced width*.

Definition 7 (induced graph, induced width). *The* induced graph *of a graphical model relative to ordering o is an undirected graph that has variables as its vertices. The edges of the graph are added by: 1) connecting all variables that are in the scope of the same function, 2) processing nodes from last to first in o, recursively connecting preceding neighbors of each node. The* induced width $w(o)$ *relative to ordering o is the maximum number of preceding neighbors across all variables in the induced graph. The* induced width of the graphical model w^* *is the minimum induced width of all orderings. It is also known as the* treewidth *of the graph.*

Theorem 1 (BE correctness and complexity). *[8] Given a reasoning task P defined over a semiring $(\boldsymbol{A}, \otimes, \oplus)$, BE is sound and complete. Given an ordering o, the* time *and* space complexity *of $BE(P)$ is exponential in the induced width of the ordering.*

Mini-bucket Elimination (MBE) [11] (see Algorithm 2) is an approximation designed to avoid the space and time complexity of BE. Consider a bucket $\mathbf{B}_i$ and an integer bounding parameter z. MBE creates a z-partition $Q = \{Q_1, ..., Q_p\}$ of $\mathbf{B}_i$, where each set $Q_j \in Q$, called *mini-bucket*, includes no more than z variables. Then, each mini-bucket is processed separately, thus computing a set of messages $\{\lambda_{ij}\}_{j=1}^{p}$, where $\lambda_{ij} = \Downarrow_{X_i} (\bigotimes_{f \in Q_j} f)$. In general, greater values of z increase the quality of the bound.

Theorem 2. *[11] Given a reasoning task P, MBE computes a bound on P. Given an integer control parameter z, the time and space complexity of MBE is exponential in z.*

3 M-best Optimization Task

In this section we formally define the problem of finding a set of best solutions over an optimization task. We consider optimization tasks defined over a set of totally or-

Algorithm 2 Mini-Bucket elimination

```
Input: A reasoning task $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \Downarrow)$; An integer parameter $z$; An ordering of variables $o = \{X_1, \dots, X_n\}$;
Output: A bound on the solution of the reasoning task.
 1: Initialize: Generate an ordered partition of functions in buckets $\mathbf{B}_1, \dots, \mathbf{B}_n$, where $\mathbf{B}_i$ contains all the functions whose highest variable in their scope is $X_i$.
 2: Backward:
 3: for $i \leftarrow n$ down to 1 do
 4:     $\{Q_1, \dots, Q_p\} \leftarrow$ Partition($\mathbf{B}_i$,$z$)
 5:     for $k \leftarrow 1$ up to $p$ do
 6:         $\lambda_{i,k} \leftarrow (\bigotimes_{f \in Q_k} f) \Downarrow_{X_i}$
 7:         Place $\lambda_{i,k}$ in the bucket $B_j$ where $j$ is the largest-index variable in $var(\lambda_{i,k})$
 8:     end for
 9: end for
10: Return: $\lambda_1$
```

dered valuations. In other words, we consider reasoning tasks where the marginalization operator $\Downarrow$ is min or max. Without loss of generality, in the following we assume maximization tasks (i.e., $\Downarrow$ is max).

Definition 8 (optimization task). *Given a graphical model $\mathcal{M}$, its optimization task is $P = (\mathcal{M}, max)$. The goal is to find a complete assignment t such that $\forall t' \in D_{\boldsymbol{X}}, C(t) \geq C(t')$. $C(t)$ is called the optimal solution.*

Definition 9 (m-best optimization task). *Given a graphical model $\mathcal{M}$, its m-best optimization task is to find m complete assignments $\boldsymbol{T} = \{t_1, \dots, t_m\}$ such that $C(t_1) >, \cdots, > C(t_m)$ and $\forall t' \in \boldsymbol{D_X} \backslash \boldsymbol{T}, \exists_{1 \leq j \leq m} t_j \; C(t') = C(t_j) \vee C(t_m) > C(t')$. The solution is the set of valuations $\{C(t_1), \dots, C(t_m)\}$, called m-best solutions.*

3.1 M-best Valuation Structure

One of the main goals of this paper is to phrase the m-best optimization task as a reasoning task over a semiring, so that well known algorithms can be immediately applied to solve this task. Namely, given an optimization task P over a graphical model $\mathcal{M}$, we need to define a reasoning task P^m that corresponds to the set of m best solutions of $\mathcal{M}$. We introduce the set of ordered m-best elements of a subset $S \subseteq \mathbf{A}$.

Definition 10 (set of ordered m-best elements). *Let S be a subset of a set of valuations $\boldsymbol{A}$. The set of ordered m-best elements of S is $Sorted^m\{S\} = \{s_1, \dots, s_j\}$, such that $s_1 > s_2 > \dots > s_j$ where $j = m$ if $|S| \geq m$ and $j = |S|$ otherwise, and $\forall s' \notin Sorted^m\{S\}, s_j > s'$.*

Definition 11 (m-space). *Let $\boldsymbol{A}$ be a set of valuations. The m-space of $\boldsymbol{A}$, noted $\boldsymbol{A}^m$, is the set of subsets of $\boldsymbol{A}$ that are sets of ordered m-best elements. Formally, $\boldsymbol{A}^m = \{S \subseteq A \mid Sorted^m\{S\} = S\}$.*

The combination and addition operators over the m-space $\mathbf{A}^m$, noted $\otimes^m$ and $sort^m$ respectively, are defined as follows.

Definition 12 (combination and addition over the m-space). *Let **A** be a set of valuations, and $\otimes$ and $\max$ be its combination and marginalization operators, respectively. Let $S, T \in \boldsymbol{A}^m$. Their* combination*, noted $S \otimes^m T$, is the set $Sorted^m\{a \otimes b \mid a \in S, b \in T\}$, while their* addition*, noted $sort^m\{S, T\}$, is the set $Sorted^m\{S \cup T\}$.*

Proposition 1. *When $m = 1$, the valuation structure $(\boldsymbol{A}^m, \otimes^m, sort^m)$ is equivalent to $(\boldsymbol{A}, \otimes, \max)$.*

Theorem 3. *The valuation structure $(\boldsymbol{A}^m, \otimes^m, sort^m)$ is a semiring.*

The theorem is proved in the Appendix A.

It is worthwhile to see the ordering defined by the semiring $(\mathbf{A}^m, \otimes^m, sort^m)$, because it would be important in the extension of Mini-Bucket Elimination (Section 5). Recall that by definition, given two elements $S, T \in \mathbf{A}^m$, $S > T$ if $S = Sorted^m\{T \cup W\}$, where $W \in \mathbf{A}^m$. We call S an *m-best bound* of T.

Definition 13 (m-best bound). *Let T, S, W be three sets of ordered m-best elements. S is an* m-best bound *of T iff $S = Sorted^m\{T \cup W\}$.*

Let us illustrate the previous definition by the following example. Let $T = \{10, 6, 4\}$, $S = \{10, 7, 4\}$, and $R = \{10, 3\}$ be three sets of ordered 3-best elements. S is not a 3-best bound of T because there is no set W such that $S = Sorted^m\{T \cup W\}$. Note that one possible set W is $S \backslash T = \{7\}$ but, $Sorted^m\{\{10, 6, 4\} \cup \{7\}\} = \{10, 7, 6\}$, which is different to S. However, S is a 3-best bound of R because $S = Sorted^m\{\{10, 3\} \cup \{7, 4\}\}$.

3.2 Vector Functions

We will refer to functions over the m-space $\mathbf{A}^m$ $f : \mathbf{D}_f \to \mathbf{A}^m$ as *vector functions*. Abusing notation, we extend the $\otimes^m$ and $sort^m$ operators to operate over vector functions similar to how operators $\otimes$ and $\oplus$ were extended to operate over scalar functions in Definition 2.

Definition 14 (combination and marginalization over vector functions). *Let $f : \boldsymbol{D}_f \to \boldsymbol{A}^m$ and $g : \boldsymbol{D}_g \to \boldsymbol{A}^m$ be two vector functions. Their combination, noted $f \overline{\bigotimes} g$, is a new function with scope $var(f) \cup var(g)$, s.t.*

$$\forall t \in \boldsymbol{D}_{var(f) \cup var(g)},\ (f \overline{\bigotimes} g)(t) = f(t) \otimes^m g(t)$$

*Let **W** $\subseteq$ **X** be a set of variables. The marginalization of f over **W**, noted $\underset{\boldsymbol{W}}{sort^m}\{f\}$, is a new function whose scope is $var(f) - $ **W**, s.t.*

$$\forall t \in D_{var(f) - \boldsymbol{W}},\ \underset{\boldsymbol{W}}{sort^m}\{f\}(t) = sort^m_{t' \in D_W}\{f(t \cdot t')\}$$

Example 3. Figure 1 shows the combination and marginalization over two vector functions h_1 and h_2 for $m = 2$ and $\otimes = \times$.

h_1: X_1	X_2	
a	a	{4,2}
a	b	{3,1}
b	a	{5}
b	b	{2}

h_2: X_2	
a	{3,1}
b	{1}

$h_1 \otimes^m h_2$: X_1	X_2	
a	a	{12,6}
a	b	{3,1}
b	a	{15,5}
b	b	{2}

$sort^m_{X_2}\{h_1\}$: X_1	
a	$\{4,3\}$
b	$\{5,2\}$

Fig. 1: Combination and marginalization over vector functions for $m = 2$ and $\otimes = \times$. For each pair of values of (X_1, X_2) the result of $h_1 \otimes^m h_2$ is an ordered set of size 2 obtained by choosing the 2 larger elements out of the result of pair-wise multiplication of the corresponding elements of h_1 and h_2. The result of $sort^m_{X_2}\{h_1\}$ is an ordered set containing the two larger values of function h_1 for each value of X_1.

3.3 M-best Optimization as a Graphical Model

The m-best extension of an optimization problem P is a new reasoning task P^m that expresses the m-best task over P.

Definition 15 (m-best extension). *Let $P = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$ be an optimization problem defined over a semiring $(\boldsymbol{A}, \otimes, \max)$. Its* m-best extension *is a new reasoning task $P^m = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}^m, \boldsymbol{F}^m, \overline{\bigotimes}, sort^m)$ over semiring $(\boldsymbol{A}^m, \otimes^m, sort^m)$. Each function $f : \boldsymbol{D}_f \rightarrow \boldsymbol{A}$ in $\boldsymbol{F}$ is trivially transformed into a new vector function $f' : \boldsymbol{D}_f \rightarrow \boldsymbol{A}^m$ defined as $f'(t) = \{f(t)\}$. In words, function outcomes of f are transformed to singleton sets in f'. Then, the set $\boldsymbol{F}^m$ contains the new f' vector functions.*

The following theorem shows that the optimum of P^m corresponds to the set of m-best valuations of P.

Theorem 4. *Let $P = (\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{A}, \boldsymbol{F}, \bigotimes, \Downarrow)$ be an optimization problem defined over semiring $(A, \otimes, \max)$ and let $\{C(t_1), \ldots, C(t_m)\}$ be its m best solutions. Let P^m be the m-best extension of P. Then, the optimization task P^m computes the set of m-best solutions of P. Formally,*

$$sort_{\boldsymbol{X}}{}^m\{\overline{\bigotimes}_{f \in \boldsymbol{F}^m} f\} = \{C(t_1), \ldots, C(t_m)\}$$

The theorem is proved in the Appendix B

It is easy to see how the same extension applies to minimization tasks. The only difference is the set of valuations selected by operator $sort^m$.

4 Bucket Elimination for the m-best Task

In this section we provide a formal description of the extension of Bucket Elimination algorithm for m best solutions based on the operators over the m-space defined in the previous section. We also provide algorithmic details for the operators and show through an example how the algorithm can be derived from first principles.

Algorithm 3 elim-m-opt algorithm

```
Input: An optimization task $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, max)$; An ordering of variables $o = \{X_1, \ldots, X_n\}$;
Output: A zero-arity function $\lambda_1 : \emptyset \rightarrow \mathbf{A}^m$ containing the solution of the $m$-best optimization task.
1: Initialize: Transform each function $f \in \mathbf{F}$ into a singleton vector function $h(t) = \{f(t)\}$;
   Generate an ordered partition of vector functions $h$ in buckets $\mathbf{B}_1, \ldots, \mathbf{B}_n$, where $\mathbf{B}_i$ contains
   all the functions whose highest variable in their scope is $X_i$.
2: Backward:
3: for $i \leftarrow n$ down to 1 do
4:     Generate $\overline{\lambda_i} = sort^m_{X_i}(\overline{\bigotimes}_{f \in B_i} f)$
5:     Generate assignment $\overline{x_i} = argsort^m_{X_i}(\overline{\bigotimes}_{f \in B_i})$, concatenate with relevant elements of
       the previously generated assignment messages.
6:     Place $\overline{\lambda_i}$ and corresponding assignments in the bucket of the largest-index variable in
       $var(\overline{\lambda_i})$
7: end for
8: Return: $\lambda_1$
```

4.1 The Algorithm Definition

Consider an optimization task $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, \max)$. Algorithm *elim-m-opt* (see Algorithm 3) is the extension of BE to solve P^m (i.e., the m-best extension of P). First, the algorithm transforms scalar functions in $\boldsymbol{F}$ to their equivalent vector functions as described in Definition 15. Then it processes the buckets from last to first as usual, using the two new combination and marginalization operators $\overline{\bigotimes}$ and $sort^m$, respectively. Roughly, the elimination of variable X_i from a vector function will produce a new vector function λ_i, such that $\lambda_i(t)$ will contain the m-best extensions of t to the eliminated variables $X_{i+1}, \ldots, X_n$ with respect to the subproblem below the bucket variable in the bucket tree. Once all variables have been eliminated, the resulting zero-arity function λ_1 contains the m-best cost extensions to all variables in the problem. In other words, λ_1 is the solution of the problem.

The correctness of the algorithm follows from the formulation of the m-best optimization task as a reasoning task over a semiring.

Theorem 5 (elim-m-opt correctness). *Algorithm* elim-m-opt *is sound and complete for finding the m best solutions over an optimization task P.*

There could be several ways to generate the set of m-best assignments, one of which is presented next and it uses the $argsort^m$ operator.

Definition 16. *Operator $argsort^m_{X_i} f$ returns a vector function $\overline{x}_i(t)$ such that $\forall t \in D_{var(f)\setminus X_i}$, where $\langle f(t \cdot {x_i}^1), \ldots, f(t \cdot {x_i}^m) \rangle$, are the m-best valuations extending t to X_i and where ${x_i}^j$ denotes the j^{th} element of $\overline{x}_i(t)$.*

In words, $\overline{x}_i(t)$ is the vector of assignments to X_i that yields the m-best extensions to t.

4.2 Illustrating the Algorithm's Derivation through an Example

We have in mind the MPE (most probable explanation) task in probabilistic networks. Consider a graphical model with four variables $\{X, Y, Z, T\}$ having the following functions (for simplicity we use un-normalizes functions). Let $m = 3$.

x	z	$f_1(z,x)$
0	0	2
0	1	2
1	0	5
1	1	1
2	0	4
2	1	3

y	z	$f_2(z,y)$
0	0	6
0	1	7
1	0	2
1	1	4
2	0	8
2	1	2

z	t	$f_3(t,z)$
0	0	1
0	1	2
1	0	4
1	1	3

Finding the m best solutions to $P(t,z,x,y) = f_3(t,z) \cdot f_1(z,x) \cdot f_2(z,y)$ can be expressed as finding Sol, defined by:

$$Sol = \underset{t,x,z,y}{sort}^{\,m} \Big(f_3(t,z) \cdot f_1(z,x) \cdot f_2(z,y) \Big) \tag{1}$$

Since operator $sort^m$ is an extension of operator max, it inherits its distributive properties over multiplication. Due to this distributivity, we can apply symbolic manipulation and migrate each of the functions to the left of the $sort^m$ operator over variables that are not in its scope. In our example we rewrite as:

$$Sol = \underset{t}{sort}^m \underset{z}{sort}^m \left(f_3(t,z) \left(\underset{x}{sort}^m f_1(z,x) \right) \left(\underset{y}{sort}^m f_2(z,y) \right) \right) \tag{2}$$

The output of $sort^m$ is a set, so in order to make (2) well defined, we replace the multipication operator by the combination over vector functions as in Definition 14.

$$Sol = \underset{t}{sort}^m \underset{z}{sort}^m (f_3(t,z) \overline{\bigotimes} (\underset{x}{sort}^m f_1(z,x)) \overline{\bigotimes} (\underset{y}{sort}^m f_2(z,y))) \tag{3}$$

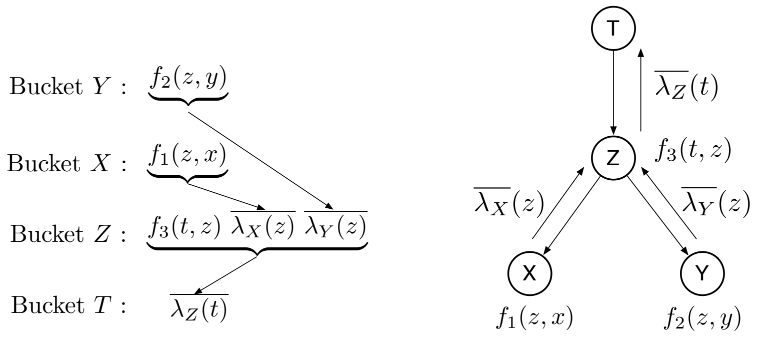
BE computes (3) from right to left, which corresponds to the elimination ordering $o = \{T, Z, X, Y\}$. We assume that the original input functions extend to vector functions, e.g., f_i is extended as $\overline{f}_i(t) = \{f_i(t)\}$. Figure 2 shows the messages passed between buckets and the bucket tree under o.

Bucket $\mathbf{B}_Y$ containing function $f_2(z,y)$ is processed first. The algorithm applies operator $\underset{y}{sort}^m$ to $f_2(z,y)$, generating a $message$, which is a vector function denoted by $\overline{\lambda_Y}(z)$, that is placed in $\mathbf{B}_Z$. Note that this message associates each z with the vector of m-best valuations of $f_2(z,y)$. Namely,

$$\underset{y}{sort}^m f_2(z,y) = (\lambda_Y^1(z), \ldots, \lambda_Y^j(z), \ldots, \lambda_Y^m(z)) = \overline{\lambda_Y}(z) \tag{4}$$

where for z each $\lambda_Y^j(z)$ is the j^{th} best value of $f_2(z,y)$. Similar computation is carried in $\mathbf{B}_X$ yielding $\overline{\lambda_X}(z)$ which is also placed in $\mathbf{B}_Z$.

z	$\overline{\lambda_X}(z)$	$\overline{x}$	$\overline{\lambda_Y}(z)$	$\overline{y}$
0	{5,4,2}	{ 1, 2, 0}	{8,6,2}	{2, 0, 1}
1	{3,2,1}	{ 2, 0, 1}	{7,4,2}	{0, 1, 2}


Bucket Y :
$\underbrace{f_2(z,y)}$
Bucket X :
$\underbrace{f_1(z,x)}$
Bucket Z :
$\underbrace{f_3(t,z)\ \overline{\lambda_X}(z)\ \overline{\lambda_Y}(z)}$
Bucket T :
$\overline{\lambda_Z}(t)$
T
Z
X
Y
$\overline{\lambda_Z}(t)$
$f_3(t,z)$
$\overline{\lambda_X}(z)$
$\overline{\lambda_Y}(z)$
$f_1(z,x)$
$f_2(z,y)$

(a) Messages passed between buckets (b) Bucket-tree

Fig. 2: Example of applying *elim-m-opt*

When processing $\mathbf{B}_Z$, we compute (see Eq. 3):

$$\overline{\lambda_Z}(t) = sort_z^m[\overline{f}_3(t,z)\overline{\bigotimes}\overline{\lambda_X}(z)\overline{\bigotimes}\overline{\lambda_Y}(z)]$$

The result is a new vector function that has m^2 elements for each tuple (t, z) as shown below.

t	z	$f_3(t,z)\overline{\bigotimes}\overline{\lambda_X}(z)\overline{\bigotimes}\overline{\lambda_Y}(z)$
0	0	{40, 32, 30, 16, 24, 12, 10, 8, 4}
0	1	{84, 56, 48, 32, 28, 24, 16, 16, 8}
1	0	{80, 64, 60, 48, 32, 24, 20, 16, 8}
1	1	{63, 42, 36, 24, 21, 18, 12, 12, 6}

Applying $sort_z^m$ to the resulting combination generates the m-best elements out of those m^2 yielding message $\overline{\lambda_Z}(t)$ along with its variable assignments:

t	$\overline{\lambda_Z}(t)$	$\langle\overline{x},\overline{y},\overline{z}\rangle$
0	{84,56,48}	$\{\langle 2,0,1\rangle, \langle 0,0,1\rangle, \langle 2,1,1\rangle\}$
1	{80,64,63}	$\{\langle 1,2,0\rangle, \langle 2,2,0\rangle, \langle 2,0,1\rangle\}$

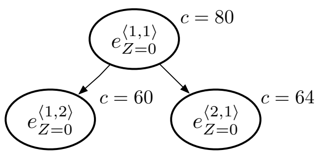
In Sect. 4.3 we show that it is possible to apply a more efficient procedure that would calculate at most $2m$ elements per tuple (t, z) instead.

Finally, processing the last bucket yields the vector of m best solution costs for the entire problem and the corresponding assignments: $Sol = \overline{\lambda_T} = \underset{t}{sort}^m\overline{\lambda_Z}(t)$ (see Fig. 2a).

$\overline{\lambda_Z}(t)$	$\langle\overline{x},\overline{y},\overline{z},\overline{t}\rangle$
{84,80,64}	$\{\langle 2,0,1,0\rangle, \langle 1,2,0,1\rangle, \langle 2,2,0,1\rangle\}$

4.3 Processing a Bucket and the Complexity of *elim-m-opt*

We will next show that the messages computed in a bucket can be obtained more efficiently than through a brute-force application of $\overline{\bigotimes}$ followed by $sort^m$. Consider


$c = 80$
$e_{Z=0}^{\langle 1,1\rangle}$
$c = 60$
$c = 64$
$e_{Z=0}^{\langle 1,2\rangle}$
$e_{Z=0}^{\langle 2,1\rangle}$

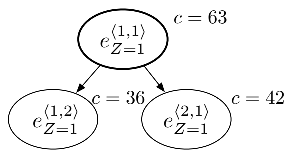

$c = 63$
$e_{Z=1}^{\langle 1,1\rangle}$
$c = 36$
$c = 42$
$e_{Z=1}^{\langle 1,2\rangle}$
$e_{Z=1}^{\langle 2,1\rangle}$

Fig. 3: The explored search space for $T = 0$ and $m = 3$. The resulting message is $\overline{\lambda_Z}(1) = \{80, 64, 63\}$.

processing $\mathbf{B}_Z$ (see Fig. 2a). A brute-force computation of

$$\overline{\lambda_Z}(t) = \underset{z}{sort}^m (f_3(z,t) \overline{\bigotimes} \overline{\lambda_Y}(z) \overline{\bigotimes} \overline{\lambda_X}(z))$$

for each t combines $f_3(z,t)$, $\overline{\lambda_Y}(z)$ and $\overline{\lambda_X}(z)$ for $\forall z \in D_Z$ first. This results in a vector function with scope $\{T, Z\}$ having m^2 elements that we call *candidate elements* and denote by $E(t,z)$. The second step is to apply $\underset{z}{sort}^m E(t,z)$ yielding the desired m best elements $\overline{\lambda_Z}(t)$.

However, since $\overline{\lambda_Y}(z)$ and $\overline{\lambda_X}(z)$ can be kept sorted, we can generate only a small subset of these m^2 candidates as follows. We denote by $e_z^{\langle i,j\rangle}(t)$ the candidate element obtained by the product of the scalar function value $f_3(t,z)$ with the i^{th} element of $\overline{\lambda_Y}(z)$ and j^{th} element of $\overline{\lambda_X}(z)$, having cost $c_z^{\langle i,j\rangle}(t) = f_3(t,z) \cdot \lambda_Y^i(z) \cdot \lambda_X^j(z)$. We would like to generate the candidates $e_z^{\langle i,j\rangle}$ in decreasing order of their costs while taking their respective indices i and j into account.

The *child elements* of $e_z^{\langle i,j\rangle}(t)$, *children*($e_z^{\langle i,j\rangle}(t)$) are obtained by replacing in the product either an element $\lambda_Y^i(z)$ with $\lambda_Y^{i+1}(z)$, or $\lambda_X^j(z)$ with $\lambda_X^{j+1}(z)$, but not both.

This leads to a forest-like search graph whose nodes are the candidate elements, where each search subspace corresponds to a different value of z denoted by $G_{Z=z}$ and rooted in $e_{Z=z}^{\langle 1,1\rangle}(t)$. Clearly, the cost along any path from a node to its descendants is non-increasing. It is easy to see that the m best elements $\overline{\lambda_Z}(t)$ can then be generated using a greedy best-first search across the forest search space $G_{Z=0} \cup G_{Z=1}$. It is easy to show that we do not need to keep more than m nodes on the OPEN list (the fringe of the search) at the same time. The general algorithm is described in Algorithm 4. The trace of the search for the elements of cost message $\overline{\lambda_Z}(t = 1)$ for our running example is shown in Figure 3.

Proposition 2 (complexity of bucket processing). *Given a bucket of a variable X over scope S having j functions $\{\overline{\lambda_1}, ..., \overline{\lambda_j}\}$ of dimension m, where m is the number of best solutions sought and k bounds the domain size, the complexity of bucket processing is $O(k^{|S|} \cdot m \cdot j \log m)$, where $|S|$ is the scope size of S.*

Proof. To generate each of the m solutions, the bucket processing routine removes the current best element from OPEN (in constant time), generates its j children and puts them on OPEN, while keeping the list sorted, which takes $O(\log(m \cdot j))$ per node, since the maximum length of OPEN is $O(m \cdot j)$. This yields time complexity of $O((m \cdot j) \cdot$

Algorithm 4 Bucket processing

```
Input: $\mathbf{B}_X$ of variable $X$ containing a set of ordered $m$-vector functions
    $\{\overline{\lambda_1}(S_1, X), \cdots, \overline{\lambda_d}(S_d, X)\}$
Output: $m$-vector function $\overline{\lambda_X}(S)$, where $S = \cup_{i=1}^{d} S_i - X$.
 1: for all $t \in D_S$ do
 2:     for all $x \in D_X$ do
 3:         $OPEN \leftarrow e_{X=x}^{\langle 1,\ldots,1 \rangle}(t)$; Sort OPEN;
 4:     end for
 5:     while $j \leq m$, by +1 do
 6:         $n \leftarrow$ first element $e_{X=x}^{\langle i_1,\cdots,i_d \rangle}(t)$ in OPEN. Remove $n$ from OPEN;
 7:         $\lambda_X^j(s) \leftarrow n$; {the $j^{th}$ element is selected}
 8:         $C \leftarrow$ children(n) = $\{e_{X=x}^{\langle i_1,\cdots,i_r+1,\cdots,i_d \rangle}(t) | r = 1..d\}$;
 9:         Insert each $c \in C$ into OPEN maintaining order based on its computed value. Check
            for duplicates; Retain the $m$ best nodes in OPEN, discard the rest.
10:     end while
11: end for
```

$log(m \cdot j))$ for all m solutions. The process needs to be repeated for each of the $O(k^{|S|})$ tuples, leading to overall complexity $O(k^{|S|} \cdot m \cdot j \log(m \cdot j))$. □

Theorem 6 (complexity of *elim-m-opt*). *Given a graphical model $(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \bigotimes)$ having n variables, whose domain size is bounded by k, an ordering o with induced-width w^* and an operator $\Downarrow = max$, the time complexity of* elim-m-opt *is $O(nk^{w^*} m \log m)$ and its space complexity is $O(mnk^{w^*})$.*

Proof. Let deg_i be the degree of the node corresponding to the variable X_i in the bucket-tree. Each bucket $\mathbf{B}_i$ contains deg_i functions and at most w^* different variables with largest domain size k. We can express the time complexity of computing a message between two buckets as $O(k^{w^*} m \cdot deg_i \log m)$ (Proposition 2), yielding the total time complexity of *elim-m-opt* of $O(\sum_{i=1}^{n} k^{w^*} m \cdot deg_i \log m)$. Assuming $deg_i \leq deg$ and since $\sum_{i=1}^{n} deg_i \leq 2n$, we get the total time complexity of $O(nmk^{w^*} \log m)$.

The space complexity is dominated by the size of the messages between buckets, each containing m costs-to-go for each of $O(k^{w^*})$ tuples. Having at most n such messages yields the total space complexity of $O(mnk^{w^*})$. □

5 Mini-Bucket Elimination for m-best Task

We next extend the *elim-m-opt* to the mini-bucket scheme. We prove that the new algorithm computes an m-best bound on the set of m-best solutions of the original problem, and describe how the m-best bound can be used to tighten the bound on the best solution of an optimization task.

5.1 The Algorithm Definition

Algorithm *mbe-m-opt* (Algorithm 5) is a straightforward extension of MBE to solve the m-best reasoning task, where the combination and marginalization operators are the

Algorithm 5 mbe-m-opt algorithm

```
Input: An optimization task $P = (\mathbf{X}, \mathbf{D}, \mathbf{A}, \mathbf{F}, \bigotimes, max)$; An ordering of variables $o = \{X_1, \ldots, X_n\}$; parameter $z$.
Output: bounds on each of the m-best solution costs and the corresponding assignments for the expanded set of variables (i.e., node duplication).
1: Initialize: Generate an ordered partition of functions $\overline{f}(t) = \{f(t)\}$ into buckets $\mathbf{B}_1, \ldots, \mathbf{B}_n$, where $\mathbf{B}_i$ contains all the functions whose highest variable in their scope is $X_i$ along $o$.
2: Backward:
3: for $i \leftarrow n$ down to 1 (Processing bucket $B_i$) do
4:     Partition functions in bucket $B_i$ into $\{Q_{i_1}, ..., Q_{i_l}\}$, where each $Q_{i_j}$ has no more than $z$ variables.
5:     Generate cost messages $\lambda_{i_j} = sort^m_{X_i}(\overline{\bigotimes}_{f \in Q_{i_j}} f)$
6:     Generate assignment using duplicate variables for each mini-bucket: $\overline{x}_{i_j} = argsort^m_{X_i}(\overline{\bigotimes}_{f \in Q_{i_j}} f)$, concatenate with relevant elements of the previously generated assignment messages
7:     Place each $\lambda_{i_j}$ and $\overline{x}_{i_j}$ in the largest index variable in $var(Q_{i_j})$
8: end for
9: Return: The set of all buckets, and the vector of m-best costs bounds in the first bucket.
```

ones defined over vector functions. The input of the algorithm is an optimization task P, and its output is a collection of bounds (i.e., an m-best bound (see Definition 13)) on the m best solutions of P.

Theorem 7 **(*mbe-m-opt* bound).** *Given a maximization task P,* mbe-m-opt *computes an m-best upper bound on the m-best optimization task P^m.*

The theorem is proved in the Appendix C

Theorem 8 **(*mbe-m-opt* complexity).** *Given a maximization task P and an integer control parameter z, the time and space complexity of* mbe-m-opt *is $O(mnk^z \log(m))$ and $O(mnk^z)$, respectively, where k is the maximum domain size and n is the number of variables.*

The theorem is proved in the Appendix D

5.2 Using the m-best Bound to Tighten the First-best Bound

Here is a simple, but quite fundamental observation: whenever upper or lower bounds are generated by solving a relaxed version of a problem, the relaxed problem's solution set contains all the solutions to the original problem. We next discuss the ramification of this observation.

Proposition 3. *Let P be an optimization problem, and let $\tilde{C} = \{\tilde{p}_1 \geq \tilde{p}_2 \geq, ..., \geq \tilde{p}_m\}$ be the m best solutions of P generated by* mbe-m-opt. *Let p^{opt} be the optimal value of P, and let j_0 be the first index such that $\tilde{p}_j = p^{opt}$, or else we assign $j_0 = m + 1$. Then, if $j_0 > m$, $\tilde{p}_m$ is an upper bound on p^{opt}, which is as tight or tighter than all other $\tilde{p}_1, ...\tilde{p}_{m-1}$. In particular $\tilde{p}_m$ is tighter than the bound $\tilde{p}_1$.*

Proof. Let $\tilde{C} = \{\tilde{p}_1 \geq \tilde{p}_2 \geq, ..., \geq \tilde{p}_{N_1}\}$ be the ordered set of valuations of all tuples over the relaxed problem (with duplicate variables). By the nature of any relaxation, $\tilde{C}$ must also contain all the probability values associated with solutions of the original problem P denoted by $C = \{p_1 \geq \cdots \geq p_{N_2}\}$. Therefore, if j_0 is the first index such that $\tilde{p}_{j_0}$ coincides with p^{opt}, then clearly for all $i < j_0$, $p^{opt} \leq \tilde{p}_i$ with $\tilde{p}_{j-1}$ being the tightest upper bound. Also, when $j_0 > m$ we have $\tilde{p}_m \geq p^{opt}$. □

In other words, if $j \leq m$, we already have the optimal value, otherwise we can use $\tilde{p}_m$ as our better upper bound. Such tighter bounds would be useful during search algorithm such as A*. It is essential therefore to decide efficiently whether a bound coincides with the exact optimal cost. Luckily, the nature of the MBE relaxation supplies us with an efficient decision scheme, since, as mentioned above, it is known that an assignment in which duplicates of variables take on identical values yields an exact solution.

Proposition 4. *Given a set of bounds produced by* mbe-m-opt $\tilde{p}_1 \geq \tilde{p}_2 \geq, ... \geq \tilde{p}_m$, *deciding if* $\tilde{p}_j = p^{opt}$ *can be done in polynomial time, more specifically in* $O(nm)$ *steps.*

Proof. *mbe-m-opt* provides both the bounds on the m-best costs and, for each bound, a corresponding tuple maintaining assignments to duplicated variables. The first assignment from these m-best bounds (going from largest to smallest) corresponding to a tuple whose duplicate variables are assigned identical values is optimal. And if no such tuple is observed, the optimal value is smaller than $\tilde{p}_m$. Since the above tests require just $O(nm)$ steps applied to m-best assignments already obtained in polytime, the claim follows. □

6 Empirical Demonstrations

We evaluated the performance of *mbe-m-opt* on four sets of instances taken from UAI 2008 competition [7] and compared our algorithm with the BMMF scheme [23].

6.1 Weighted Constraint Satisfaction Problems

The first part of our empirical evaluation assumed solving the Weighted CSP task, i.e, summation-minimization problem. We ran *mbe-m-opt* on 20 WCSP instances using z-bound equal to 10 and number of solutions m equal to 10. Table 1 shows for each instance the time in seconds it took *mbe-m-opt* to solve the 10-best problem and the values of the lower bounds on each of the first ten best solutions. For each problem instance we also show the number of variables n, the largest domain size k and the induced width w^*. Note that 9 of the instances have induced width less than the z-bound=10 and thus are solved exactly. We see that as the index number of solution goes up, the value of the corresponding lower bound increases, getting closer to the exact best solution. This demonstrates that there is a potential of improving the bound on the optimal assignment using the m-best bounds as discussed in Sect 5.2. Figure 4 illustrates this observation in graphical form, showing the dependency of the lower bounds on the solution index number for selected instances.

Instance)	n	k	w^*	time (sec)	Solution index number									
					1	2	3	4	5	6	7	8	9	10
1502.uai	209	4	6	0.11	228.955109	228.9552	228.955292	228.955414	229.053192	229.053284	229.053406	229.053497	229.141693	229.141785
29.uai	82	4	14	55.17	147.556778	147.557236	147.924484	147.924942	148.188965	148.189423	148.556671	148.557129	148.924393	148.92485
404.uai	100	4	19	3.96	147.056229	148.001511	148.056122	149.001404	149.056015	149.05603	150.001297	150.001312	150.055923	151.001205
408.uai	200	4	35	80.27	436.551117	437.17923	437.549042	437.550018	437.550995	438.177155	438.178131	438.179108	438.547943	438.549896
42.uai	190	4	26	61.16	219.98053	219.980713	220.014938	220.015106	220.048157	220.048325	220.08255	220.082733	220.912811	220.912827
503.uai	143	4	9	3.58	225.038483	225.039368	225.039398	225.040283	226.037476	226.037918	226.037933	226.038361	226.038376	226.038391
GEOM30a_3.uai	30	3	6	0.03	0.008100	1.008000	2.007898	2.007899	2.007899	3.007798	3.007798	3.007798	3.007799	4.007700
GEOM30a_4.uai	30	4	6	0.19	0.008100	1.008000	2.007899	2.007899	3.007799	3.007799	4.007701	4.007701	4.007702	5.007601
GEOM30a_5.uai	30	5	6	0.84	0.008100	1.008000	2.007898	2.007899	2.007899	3.007798	3.007798	3.007798	3.007799	4.007700
GEOM40_2.uai	40	2	5	0	0.007800	2.007599	2.007599	2.007600	3.007499	3.007499	3.007500	4.007398	4.007399	4.007400
GEOM40_3.uai	40	3	5	0.01	0.007800	2.007599	2.007599	2.007600	3.007500	4.007399	4.007399	4.0074	4.007400	4.007401
GEOM40_4.uai	40	4	5	0.11	0.007800	2.007598	2.007599	2.007599	2.007599	2.007599	3.007499	3.007499	3.007500	4.007400
GEOM40_5.uai	40	5	5	0.16	0.007800	2.007599	2.007600	2.007600	3.007500	4.007399	4.007400	4.007400	4.007401	4.007401
le450_5a_2.uai	450	2	293	6.06	0.571400	1.571300	1.571300	1.571300	1.571300	1.571300	1.571300	2.571198	2.571199	20.569397
myciel5g_3.uai	47	3	19	6.39	0.023600	1.023500	1.023500	2.023399	3.023299	4.023202	10.022601	11.022501	11.022502	11.022503
myciel5g_4.uai	47	4	19	129.54	0.023600	1.023500	1.023500	2.023397	2.023398	2.023398	2.023398	2.023399	3.023297	3.023298
queen5_5_3.uai	25	3	18	5.53	0.01600	1.015900	1.015901	2.015797	2.015797	2.015798	3.015694	3.015696	3.015697	3.015697
queen5_5_4.uai	25	4	18	122.26	0.01600	1.015900	1.015900	1.015901	1.015901	2.015796	2.015797	2.015797	2.015797	2.015790

Table 1: The lower bounds on the 10 best solutions found by *mbe-m-opt* ran with z-bound=10 and $m = 10$. We also report the runtime in seconds, number of variables n, induced width w^* and largest domain size k.

6.2 Most Probable Explanation Problems

For the second part of the evaluation the *mbe-m-opt* was solving the MPE problem, i.e. max-product task on three sets of instances: pedigrees, grids and mastermind. We search for $m \in [1, 5, 10, 20, 50, 100, 200]$ solutions with z-bound equal to 10.

Pedigrees. The set of pedigrees contains 15 instances with several hundred variables and induced width from 15 to 30. Table 2 contains the runtimes in seconds for each of the number of solutions m along with the parameters of the problems. Figure 5 presents the runtime in seconds against the number of solutions m for chosen pedigrees. Figure 6 demonstrates the difference between the way the runtime would scale according to the theoretical worst case peformance analysis and the empirical runtimes obtained for various values of m. For three chosen instances we plot the experimental runtimes in seconds against the number of solutions m and the theoretical curve obtained by multiplying the value of empirical runtime for $m = 1$ by the factor of $m \log m$ for m equal to 5, 10, 50, 100 and 200. We see that the empirical curve lays much lower than theoretical for all instances.

Figure 7 illustrates the potential usefulness of the upper bounds on m best solutions as an approximation of the best solution. We plot in logarithmic scale the values of upper bounds on the 100 best solutions found by *mbe-m-opt* for the z-bounds ranging from 10 to 15. When using MBE as an approximation scheme, the common rule of thumb is to run the algorithm with the highest z-bound possible. In general, higher z-bound indeed corresponds to better accuracy, however increasing the parameter by a small amount (one or two) does not provably produce better results, as we can see in our example, where *mbe-m-opt* with z-bound=10 achieves better accuracy then the ones with z-bound=11 and z-bound=12. Such behaviour can be explained by the differences in partitioning of the buckets into mini-buckets due to the changing of the control parameter z, which greatly influences the accuracy of MBE results. On the other hand, the upper bound on each next solution is always at least as good as the previous one, thus increase in m never leads to a worse bound and possibly will produce a better one.

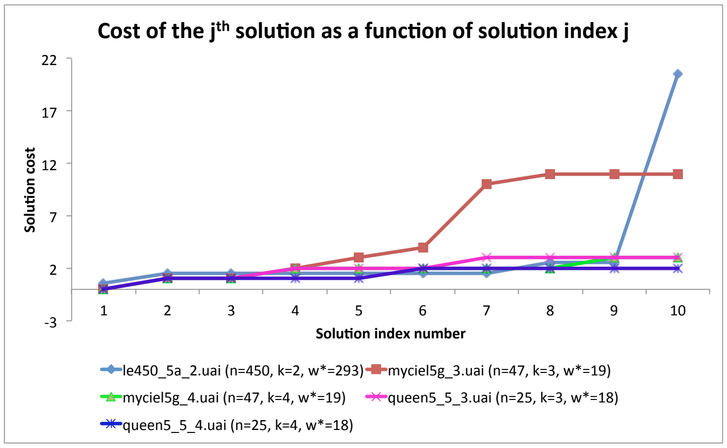


Fig. 4: The change in the cost of the j^{th} solution as j increases for chosen WCSP instances. Results are obtained by *mbe-m-opt* with z-bound=10.

However, we acknowledge that the power of *mbe-m-opt* with larger m for improving the upper bound on the 1^{st} best solution is quite weak compared with using higher z-bound. Although theory suggests that the time and memory complexity of *mbe-m-opt* is exponential in the parameter z, while only depending as a factor of $m \log m$ on the number of solutions, our experiments show that in order to obtain a substantial improvement of the bound it might be necessary to use high values of m. For example, for a problem with binary variables *mbe-m-opt* with $m = 1$ and a certain z-bound z is equivalent in terms of complexity to *mbe-m-opt* with $m = 3$ and z-bound $(z - 1)$. We observed that the costs of the first and third solutions are quite close for the instances we considered. In order to characterize when the use of *mbe-m-opt* with higher m would add power over increasing the z-bound the study of additional classes of instances is required.

Grids. The set of grids contains 30 instances with 100 to 2500 binary variables and tree-width from 12 to 50. The parameters of each instance can be seen in Table 3 that contains the runtimes in seconds for each value of number of solutions m. Theory suggests that the runtimes for $m = 1$ and $m = 100$ should differ by at least two orders of magnitude, however, we can see that in practice *mbe-m-opt* scales much better. Figure 8 shows graphically the dependency of the runtime in seconds on the number of solutions m for 10 selected instances.

Mastermind. The mastermind set contains 15 instances with several thousand binary variables and tree-width ranging from 19 to 37. The instances parameters can be seen in Table 4, that shows how the run time changes with various numbers of best solutions m. We refrain from reporting and discussing the values of the upper bounds found, since mastermind instances in question typically have a large set of solutions with the same costs, making the values of the bounds not particular informative.

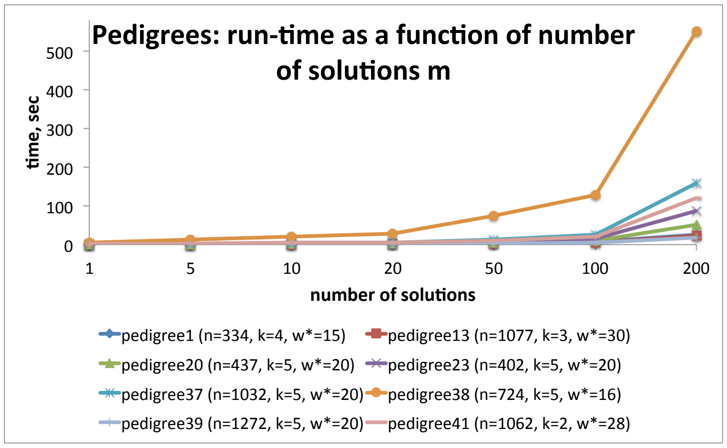


Fig. 5: The run time (sec) for pedigree instances as a function of number of solutions m. *mbe-m-opt* ran with the z-bound=10.

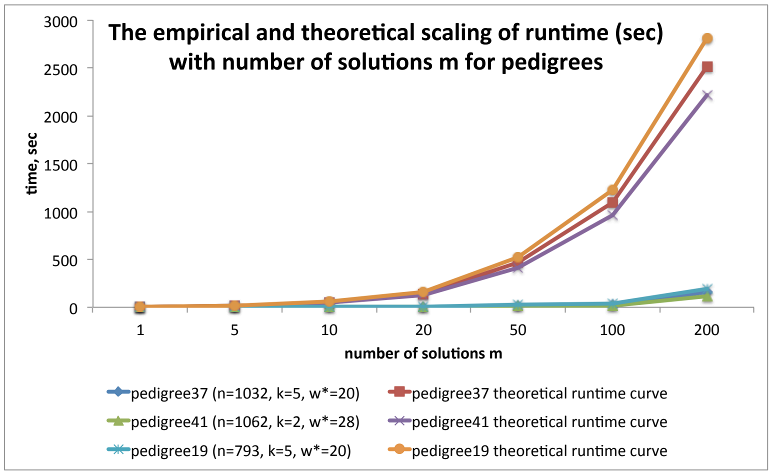


Fig. 6: The empirical and theoretical runtime scaling with number of solutions m for chosen pedigree instances. The theoretical curve is obtained by multiplying the experimental runtime in seconds obtained for $m = 1$ by the factor of $m \log m$ for values $m = 5, 10, 20, 50, 100, 200$.

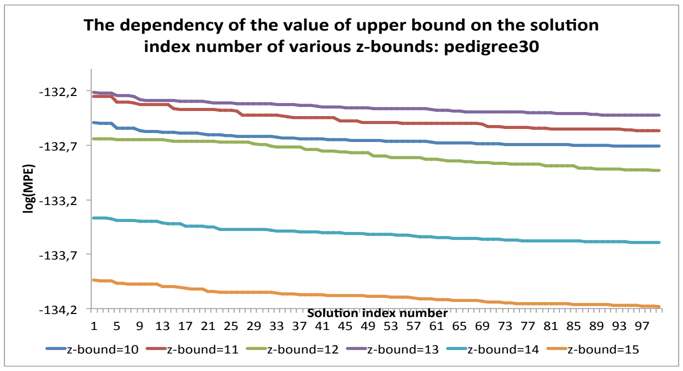


Fig. 7: The upper bounds on the 100 best solutions (in log scale) found by *mbe-m-opt* ran with z-bounds$\in [10, 11, 12, 13, 14, 15]$ for pedigree30 instance. The parameters of the problem: n=1289, k=5, w^*=20.

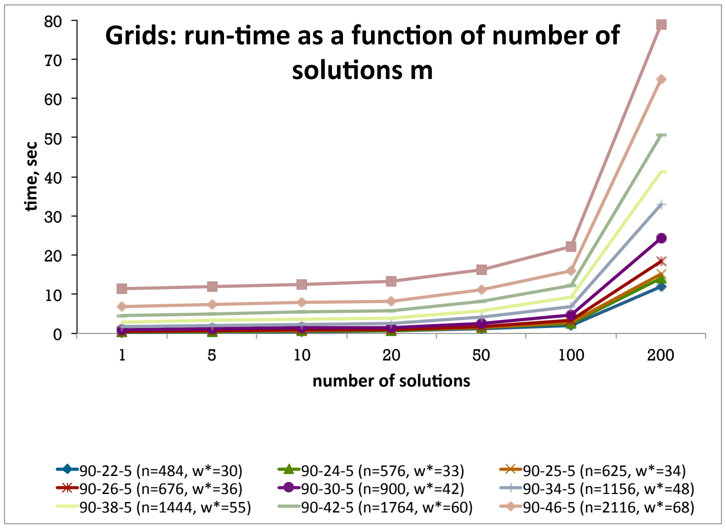


Fig. 8: Selected binary grid instances: *mbe-m-opt* run time (sec) as a function of number of solutions m. The z-bound=10.

Instances	n	k	w^*	Runtime (sec)						
				m=1	m=5	m=10	m=20	m=50	m=100	m=200
pedigree1	334	4	15	0.22	0.57	1.01	1.46	3.35	6.87	25.46
pedigree13	1077	3	30	0.64	1.06	1.32	1.65	2.77	5.06	23.80
pedigree19	793	3	21	1.84	4.67	7.65	10.17	24.12	44.17	194.79
pedigree20	437	5	20	0.54	1.22	1.83	2.34	5.00	9.43	50.36
pedigree23	402	5	20	0.92	2.09	2.89	3.58	7.51	14.63	87.22
pedigree30	1289	5	20	0.38	0.66	1.00	1.26	2.48	4.58	19.53
pedigree31	1183	5	28	0.83	1.82	2.68	3.60	7.65	13.16	57.35
pedigree33	798	4	24	0.38	0.76	1.11	1.23	2.60	4.72	27.81
pedigree37	1032	5	20	1.64	3.27	4.56	6.25	14.15	26.43	158.74
pedigree38	724	5	16	4.52	11.77	19.63	28.87	73.21	127.65	552.30
pedigree39	1272	5	20	0.33	0.63	0.89	1.25	2.42	4.64	18.31
pedigree41	1062	5	28	1.45	3.33	4.43	5.56	11.67	20.59	120.79
pedigree51	871	5	39	0.76	1.24	1.65	2.16	3.98	6.97	33.95
pedigree7	867	4	32	0.66	1.17	1.61	2.15	4.45	8.01	39.26
pedigree9	935	7	27	0.85	1.48	2.12	2.77	5.70	9.49	50.58

Table 2: Runtime (sec) of *mbe-m-opt* on pedigree instances searching for the following number of solutions: $m = \in [1, 5, 10, 20, 50, 100, 200]$ with the z-bound=10. We report the number of variables n, largest domain size k and induced width w^*.

Instances	n	w^*	Runtime (sec)						
			m=1	m=5	m=10	m=20	m=50	m=100	m=200
50-15-5	144	15	0.07	0.15	0.22	0.30	0.68	1.27	5.68
50-16-5	256	21	0.07	0.17	0.25	0.33	0.68	1.34	6.30
50-17-5	289	22	0.11	0.24	0.33	0.45	1.00	1.87	8.70
50-18-5	324	24	0.14	0.29	0.35	0.52	1.05	2.04	9.13
50-19-5	361	25	0.13	0.29	0.41	0.54	1.15	2.24	9.87
50-20-5	400	27	0.18	0.33	0.44	0.59	1.20	2.28	10.65
75-16-5	256	21	0.08	0.17	0.21	0.27	0.56	1.09	5.92
75-17-5	289	22	0.10	0.21	0.27	0.36	0.75	1.46	7.83
75-18-5	324	24	0.12	0.23	0.30	0.40	0.79	1.58	8.34
75-19-5	361	25	0.14	0.26	0.34	0.47	0.94	1.86	9.22
75-20-5	400	27	0.18	0.30	0.38	0.52	0.97	1.80	9.78
75-21-5	441	28	0.20	0.36	0.44	0.60	1.07	2.03	10.91
75-22-5	484	30	0.25	0.40	0.53	0.68	1.28	2.49	12.40
75-23-5	529	31	0.29	0.47	0.56	0.71	1.36	2.44	13.11
75-24-5	576	32	0.34	0.51	0.65	0.81	1.49	2.87	14.58
75-25-5	625	34	0.41	0.62	0.74	0.93	1.71	3.18	16.08
75-26-5	676	36	0.49	0.73	0.90	1.17	2.06	3.86	19.06
90-20-5	400	27	0.17	0.27	0.35	0.44	0.81	1.57	9.26
90-21-5	441	28	0.02	0.35	0.41	0.52	0.97	1.91	10.72
90-22-5	484	30	0.25	0.41	0.47	0.61	1.10	2.08	11.85
90-23-5	529	31	0.29	0.46	0.55	0.66	1.17	2.27	12.63
90-24-5	576	33	0.34	0.49	0.60	0.74	1.36	2.61	13.98
90-25-5	625	34	0.42	0.58	0.70	0.83	1.50	2.80	15.26
90-26-5	676	36	0.49	0.71	0.85	1.01	1.87	3.42	18.36
90-30-5	900	42	0.93	1.25	1.40	1.59	2.60	4.62	24.26
90-34-5	1156	48	1.69	2.07	2.29	2.60	4.15	6.77	32.93
90-38-5	1444	55	2.86	3.26	3.57	3.98	5.72	9.27	41.33
90-42-5	1764	60	4.57	5.10	5.49	5.88	8.32	12.31	50.70
90-46-5	2116	68	6.81	7.42	7.97	8.33	11.09	16.06	64.88
90-50-5	2500	74	11.3	12.07	12.51	13.2	16.25	22.09	78.70

Table 3: Binary grid instances: runtime (sec) of *mbe-m-opt* for the number of required solutions $m \in [1, 5, 10, 20, 50, 100, 200]$ with the z-bound=10. We report the number of variables n and induced width w^*.

6.3 Comparison with BMMF.

BMMF [23] is a Belief Propagation based algorithm which is exact when ran on junction trees and approximate if the problem graph has loops. We compared the performance of *mbe-m-opt* and BMMF on randomly generated 10 by 10 binary grids. The algorithms differ in the nature of the outputs: BMMF provides approximate solutions with

Instances	n	k	w^*	Runtime (sec)						
				m=1	m=5	m=10	m=20	m=50	m=100	m=200
mastermind_03_08_03-0006	1220	2	19	0.44	0.57	0.64	0.72	1.03	2.68	13.11
mastermind_03_08_03-0007	1220	2	18	0.25	0.33	0.35	0.40	0.67	1.81	8.90
mastermind_03_08_03-0014	1220	2	20	0.68	0.84	0.92	0.98	1.51	3.76	17.77
mastermind_03_08_04-0004	2288	2	30	1.98	2.25	2.28	2.49	3.42	7.11	32.12
mastermind_03_08_04-0005	2288	2	30	1.92	2.20	2.35	2.50	3.44	7.16	32.08
mastermind_03_08_04-0010	2288	2	29	2.53	2.82	2.97	3.09	4.17	8.25	34.89
mastermind_03_08_04-0011	2288	2	29	3.55	3.85	4.00	4.16	5.40	9.90	38.48
mastermind_03_08_05-0001	3692	2	37	6.33	6.73	7.02	7.24	9.10	15.83	59.03
mastermind_03_08_05-0005	3692	2	37	6.33	6.85	7.04	7.29	9.08	15.6	58.77
mastermind_03_08_05-0009	3692	2	37	3.44	3.72	3.81	3.97	5.18	9.59	38.62
mastermind_03_08_05-0010	3692	2	37	6.23	6.57	6.90	7.10	8.80	14.87	56.43
mastermind_04_08_03-0000	1418	2	24	1.12	1.30	1.41	1.49	2.16	4.51	20.33
mastermind_04_08_03-0013	1418	2	23	1.12	1.33	1.43	1.51	2.19	4.60	20.73
mastermind_05_08_03-0004	1616	2	27	1.22	1.43	1.47	1.57	2.22	4.60	20.39
mastermind_05_08_03-0006	1616	2	27	0.21	0.23	0.25	0.26	0.37	0.82	3.84

Table 4: The runtime (sec) of *mbe-m-opt* for the mastermind instances. Number of required solutions $m \in [1, 5, 10, 20, 50, 100, 200]$, z-bound=10. We report the number of variables n, induced width w^*, domain size k.

no guarantees while *mbe-m-opt* generates bounds on all the m-best solutions. Moreover, the runtimes of the algorithms are not comparable since our algorithm is implemented in C and BMMF in Matlab, which is inherently slower. For most instances that *mbe-m-opt* can solve exactly in under a second, BMMF takes more than 5 minutes.

Still, some information can be learned from viewing the two algorithms side by side as is demonstrated by typical results in Figure 10. For two chosen instances we plot the values of the 10-best bounds outputted by both algorithms in logarithmic scale as a function of the solution index. We also show the exact solutions found by the algorithm *elim-m-opt*. We can see that *mbe-m-opt* with the z-bound equal to 10 can produce upper bounds that are considerably closer to the exact solutions than the results outputted by BMMF. Admittedly, these experiments are quite preliminary and not conclusive.

7 Related Work: the m-best Algorithms

The previous works on finding the m best solutions describe either exact or approximate schemes. The exact algorithms can be characterized as inference based or search based, as we elaborate next.

Earlier exact inference schemes. As mentioned above, one of the most influential works among the algorithms solving the m-best combinatorial optimization problems is the widely applicable iterative scheme developed by Lawler[18]. Given a problem with n variables, the main idea of Lawler's approach is to find the best solution first and then to formulate n new problems that exclude the best solution found, but include all others. Each one of the new problems is solved optimally yielding n candidate solutions, the best among which becomes the overall second best solution. The procedure is repeated until m best solutions are found. The complexity of the algorithm is $O(nmT(n))$, where $T(n)$ is the complexity of finding a single best solution.

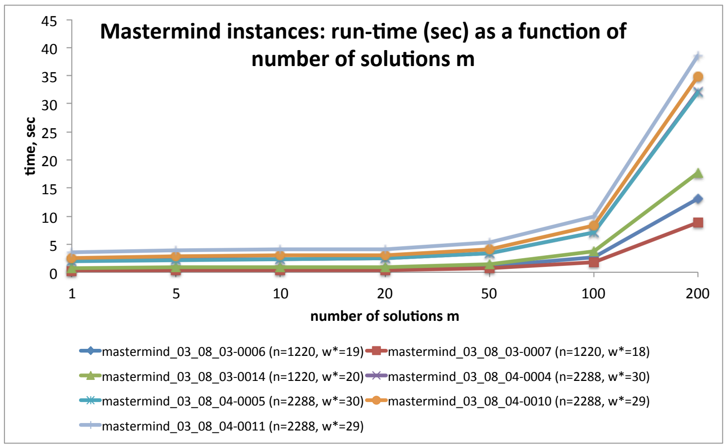


Fig. 9: *mbe-m-opt* run time (sec) as a function of number of solutions m for the mastermind instances. The z-bound=10.

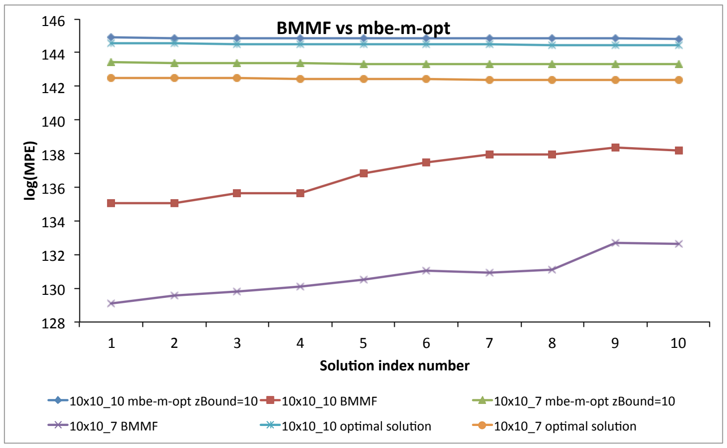


Fig. 10: Comparison of *mbe-m-opt* with z-bounds 10 and BMMF on random 10x10 grids. The exact solutions obtained by *elim-m-opt*. The *mbe-m-opt* provides upper bounds on the solutions, BMMF gives no guarantees whether it outputs an upper or a lower bound. In this particular example BMMF outputs lower bounds on the exact solutions.

Hamacher and Queyranne [15] built upon Lawler's work and presented a method that assumes the ability to directly find both the best and second best solutions to a problem. After finding the two best solutions, a new problem is formulated, so that the second best solution to the original problem is the best solution to the new one. The second best solution for the new problem is found, to become the overall third best solution and the procedure is repeated until all m solutions are found. The time complexity of the algorithm is $O(m \cdot T_2(n))$, where $T_2(n)$ is the complexity of finding the second best solution to the problem. The complexity of this method is always bounded from above by that of Lawler, seeing as Lawler's scheme can be used as an algorithm for finding the second best task.

Lawler's approach was applied by Nilsson to a join-tree [19]. Unlike Lawler or Hamacher and Queyranne, who are solving n problems from scratch in each iteration of the algorithm, Nilsson is able to utilize the results of previous computations for solving newly formulated problems. The worst case complexity of the algorithm is $O(m \cdot T(n))$, where $T(n)$ is the complexity of finding a single solution by the max-flow algorithm. If applied to a bucket-tree, Nilsson's algorithm has run time of $O(nk^{w^*} + mn\log(mn) + mnk)$.

More recently Yanover and Weiss [23] extended Nilsson's idea for the max-product Belief Propagation algorithm, yielding a belief propagation approximation scheme for loopy graphs, called BMMF, which also finds solutions iteratively and which we compared against. At each iteration BMMF uses loopy Belief Propagation to solve two new problems obtained by restricting the values of certain variables. When applied to junction tree, it can function as an exact algorithm with complexity $O(mnk^{w^*})$.

Two algorithms based on dynamic programming, similar to elim-m-opt, were developed by Serrousi and Golmard [21] and Elliot [12]. Unlike the previously mentioned works, Seroussi and Golmard don't find solutions iteratively from 1^{st} down to m^{th}, but extracts the m solutions directly, by propagating the m best partial solutions along a junction tree that is pre-compiled. Given a junction tree with p cliques, each having at most deg children, the complexity of the algorithm is $O(m^2 p \cdot k^{w^*} deg)$. Nilsson showed that for most problem configurations his algorithm is superior to the one by Seroussi and Golmard.

Elliot [12] explored the representation of Valued And-Or Acyclic Graph, i.e., smooth deterministic decomposable negation normal form (sd-DNNF) [6]. He propagates the m best partial assignments to the problem variables along the DNNF structure which is pre-compiled as well. The complexity of Elliot's algorithm is $O(nk^{w^*} m \log(m \cdot deg))$, excluding the cost of constructing the sd-DNNF.

Earlier search schemes. The task of finding m best solutions is closely related to the problem of k shortest paths (KSP) which usually is solved using search. It is known that many optimization problems can be transformed into problems of finding a path in a graph. For example, the task of finding the lowest cost solution to a weighted constraint satisfaction problem can be represented as a search for a shortest path in a graph, whose vertices correspond to assignments of the original problem variables and the lengths of edges are chosen according to the cost functions of the constraint problem. A good survey of different k shortest path algorithms can be found in [5] and [13]. The majority of the algorithms developed for solving KSP assume that the entire search graph is available as an input and thus are not directly applicable to the tasks formulated over graphical models, since for most of them storing the search graph explicitely is infeasible. One very recent exception is the work by Aljazzar and Leue [2]. Their method, called K^*, finds the k shortest paths while generating the search graph ”on-the-fly” and thus can be potentially useful for solving problems defined over graphical models. Assuming application to an AND/OR search graph [10] and given a consistent heuristic, K^* yields asymptotic worst-case time and space complexity of $O(n \cdot k^{w^*} \cdot w^* \log(nk) + m)$, thus displaying the best scaling with the required number of solutions m compared to all schemes mentioned here.

¡¡¡¡¡¡¡ .mine In recent paper [9] we proposed two new algorithms, m-A* and m-$B\&B$, that extend best first and branch and bound search respectively to finding the m best solutions, and their modifications for graphical models: m-$AOBF$ and m-$AOBB$. We showed that m-A^* is optimally efficient compared to any other algorithm that searches the same search space using the same heuristic function. The theoretical worse case time complexity for m-$AOBF$ is $O(n \cdot m \cdot k^{w^*})$ and for m-$AOBB$ is $O(n \cdot deg \cdot m \log mk^{w^*})$. However, we showed that the worst case analysis does not provide an accurate picture of algorithms’ performance and in practice in most cases they are considerably more efficient. ======= In a recent paper [9] we proposed two new algorithms, m-A* and m-$B\&B$, that extend best first and branch and bound search respectively to finding the m-best solutions, and their modifications for graphical models: m-$AOBF$ and m-$AOBB$. We showed that m-A^* is optimally efficient compared to any other algorithm that searches the same search space using the same heuristic function. The theoretical worse case time complexity for m-$AOBF$ is $O(n \cdot m \cdot k^{w^*})$ and for m-$AOBB$ is $O(n \cdot deg \cdot m \log mk^{w^*})$. However, we showed that the worst case analysis does not provide an accurate picture of algorithms’ performance and in practice in most cases they are considerably more efficient. ¿¿¿¿¿¿¿ .r231

We also presented *BE-Greedy-m-BF*, a hybrid of variable elimination and best first search scheme. The idea behind the method is to use Bucket Elimination algorithm to calculate the costs of the best path from any node to the goal and use this information as an exact heuristic for A* search. *BE-Greedy-m-BF* has the time and space complexity of $O(nk^{w^*} + nm)$ and, unlike K^*, our scheme does not require complex data structures or precomputed heuristics.

Earlier approximation schemes. In addition to BMMF, another extension of Nilsson’s and Lawler’s idea that yields an approximation scheme is an algorithm called STRIPES by [14]. They focus on m-MAP problem over binary Markov networks, solving each new subproblem by an LP relaxation. The algorithm solves the task exactly if the solutions to all LP relaxations are integral, and provides an upper bound of each m MAP assignments otherwise. In contrast, our algorithm *mbe-m-opt* can compute bounds over any graphical model (not only binary) and over a variety of m-best optimization tasks.

Other related works. Very recently, [4] studied the computational complexity of computing the next solution in some graphical models such as constraint and preference-based networks. They showed that the complexity of this task depends on the structure of the graphical model and on the strict order imposed over its solutions. It is easy to see that our m-best task can be solved by iteratively finding the next solution until m solutions with different valuation have been found. However, since our m-best task defines a partial order over solutions and it only considers solutions with different valuation, further study is needed to determine if the tractability of our problem is the same as that of the problem of finding the next solution.

8 Conclusions

We presented a formulation of the m-best reasoning task within a framework of semiring, thus making all existing inference and search algorithms immediately applicable for the task via the definition of the combination and elimination operators. We then focused on inference algorithms and provided a bucket elimination algorithm, *elim-m-opt*, for the task. Analysis of the algorithm's performance and relation with earlier work is provided.

We emphasize that the practical significance of the algorithm is primarily for approximation through the mini-bucket scheme, since other exact schemes have better worst-case performance.

Furthermore, it could also lead to loopy propagation message-passing schemes that are highly popular for approximations in graphical models. For example, *elim-m-opt* can be extended into a loopy max-prod for the m-best task, which would differ from the scheme approach by Yanover and Weiss that uses loopy max-prod for solving a sequence of optimization problems in the style of Lawler's approach.

Our empirical analysis demonstrates that *mbe-m-opt* scales as a function of m better than worst-case analysis predict. Comparison with other exact and approximation algorithms is left for future work.

Acknowledgments This work was partially supported by NSF grants IIS-0713118 and IIS-1065618, NIH grant 5R01HG004175-03, and Spanish CECyT project TIN2009-13591-C02-0.

A Proof of Theorem 3

Let S, T, R be arbitrary elements of $\mathbf{A}^m$. We prove one by one the required conditions.

- commutativity of $\otimes^m$. By definition, $S \otimes^m T = Sorted^m\{a \otimes b \mid a \in S, b \in T\}$. Since $\otimes$ is commutative, the previous expression is equal to $Sorted^m\{b \otimes a \mid b \in T, a \in S\} = T \otimes^m S$.
- associativity of $\otimes^m$. We have to prove that $(S \otimes^m T) \otimes^m R = S \otimes^m (T \otimes^m R)$. Suppose that the previous equality does not hold. Then, it would imply that:
  i. there may exist an element $a \in (S \otimes^m T) \otimes^m R$, s.t. $a \notin S \otimes^m (T \otimes^m R)$; or,
  ii. there may exist an element $a \in S \otimes^m (T \otimes^m R)$, s.t. $a \notin (S \otimes^m T) \otimes^m R$.

  We show that both cases are impossible.
  Consider the first case. Let $\{a_1, \ldots, a_m\} = S \otimes^m (T \otimes^m R)$ where $\forall_{1 \leq i < m}, a_i > a_m$. Since $a \notin S \otimes^m (T \otimes^m R)$, it means that $a_m > a$. Element a comes from the

combination of three elements $a = (s \otimes t) \otimes r$. Each element a_i comes from the combination of three elements $a_i = s_{a_i} \otimes (t_{a_i} \otimes r_{a_i})$. By associativity of operator $\otimes$, $a_i = (s_{a_i} \otimes t_{a_i}) \otimes r_{a_i}$. Then,

- If $\forall_{1 \leq i \leq m}, s_{a_i} \otimes t_{a_i} \in S \otimes^m T$, then $(s_{a_i} \otimes t_{a_i}) \otimes r_{a_i} > (s \otimes t) \otimes r$ for all $1 \leq i \leq m$, and $a \notin (S \otimes^m T) \otimes^m R$, which contradicts the hypothesis.
- If $\exists_{1 \leq j \leq m}, s_{a_j} \otimes t_{a_j} \notin S \otimes^m T$, then there exists an element $s' \otimes t' > s_{a_j} \otimes t_{a_j}$. By monotonicity of $>$, $(s' \otimes t') \otimes r_{a_j} > (s_{a_j} \otimes t_{a_j}) \otimes r_{a_j}$. As a consequence $(s_{a_m} \otimes t_{a_m}) \otimes r_{a_m} \notin (S \otimes^m T) \otimes^m R$. Since $a_m > a$, then $a \notin (S \otimes^m T) \otimes^m R$, which contradicts the hypothesis.

The proof for the second case is the same as above, but interchanging the role of a and $\{a_1, \ldots, a_m\}$, and S and R.

– commutativity of $sort^m$. By definition, $sort^m\{S, T\} = Sorted^m\{S \cup T\}$. Since set union is commutative, $Sorted^m\{S \cup T\} = Sorted^m\{T \cup S\}$ which is by definition $sort^m\{T, S\}$.

– associativity of $sort^m$. By definition, $sort^m\{sort^m\{S, T\}, R\} = Sorted^m\{Sorted^m\{S \cup T\} \cup R\}$, and $sort^m\{S, sort^m\{T, R\}\} = Sorted^m\{S \cup Sorted^m\{T \cup R\}\}$. Clearly, the two expressions are equivalent to $Sorted^m\{S \cup T \cup R\}$.

– $\otimes^m$ distributes over $sort^m$. Let us proceed by induction:

1. Base case. When $m = 1$, by Proposition 1, the valuation structure $(\mathbf{A}^m, \otimes^m, sort^m)$ is a semiring and, as a consequence, $\otimes^m$ distributes over $sort^m$.
2. Inductive step. Up to m, operator $\otimes^m$ distributes over $sort^m$, and let $\{a_1, \ldots, a_m\}$ be its result. We have to prove that $S \otimes^{m+1} (sort^{m+1}\{T, R\}) = sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$. By definition of the operators, the result is the same ordered set of elements $\{a_1, \ldots, a_m\}$ plus one element a_{m+1}. Suppose that $\otimes^{m+1}$ does not distribute over $sort^{m+1}$. Then, it would imply that:
   i. Element $a_{m+1} \in S \otimes^{m+1} (sort^{m+1}\{T, R\})$, but $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$; or,
   ii. Element $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, but $a_{m+1} \in sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$

   We show that both cases are impossible.
   Consider the first case. Since $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, it means that $\exists a' \in sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$ such that $a' > a_{m+1}$. Element a' comes from the combination of two elements $a' = s' \otimes u'$, where $s' \in S$ and $u' \in T$ or $u' \in R$. Then:
   - If $u' \in sort^{m+1}\{T, R\}$, then since $a' > a_{m+1}$, by definition of $\otimes^{m+1}$, $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, which contradicts the hypothesis.
   - If $u' \notin sort^{m+1}\{T, R\}$, then $\exists u'' \in sort^{m+1}\{T, R\}$ such that $u'' > u'$. By monotonicity of the order, $u'' \otimes s' > u' \otimes s'$ and, by transitivity, $u'' \otimes s' > a_{m+1}$. By definition of $\otimes^{m+1}$, $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, which contradicts the hypothesis.

   Consider now the second case. Since $a_{m+1} \notin S \otimes^{m+1} (sort^{m+1}\{T, R\})$, it means that $\exists a' \in S \otimes^{m+1} (sort^{m+1}\{T, R\})$ such that $a' > a_{m+1}$. Element a' comes from the combination of two elements $a' = s' \otimes u'$, where $s' \in S$ and $u' \in T$ or $u' \in R$. Then:
   - If $u' \in T$:

  - and $a' \in S \otimes^{m+1} T$. If $a \in S \otimes^{m+1} T$, since $a' > a_{m+1}$ and by definition of $\otimes^{m+1}$, $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, which contradicts the hypothesis. If $a \notin S \otimes^{m+1} T$, since $a' > a_{m+1}$ and by definition of $sort^{m+1}$, $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, which contradicts the hypothesis.
  - and $a' \notin S \otimes^{m+1} T$. Then, $\exists a'' \in S \otimes^{m+1} T$ such that $a'' > a'$. By transitivity of the order, $a'' > a_{m+1}$. Then, either by definition of $\otimes^{m+1}$ or by definition of $sort^{m+1}$, $a_{m+1} \notin sort^{m+1}\{S \otimes^{m+1} T, S \otimes^{m+1} R\}$, which contradicts the hypothesis.
- If $u' \in R$. The reasoning is the same as above, but interchanging the role of T and R.

$\square$

B Proof of Theorem 4

By definition of $sort^m$,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} (\overline{\bigotimes}_{f \in \mathbf{F}^m} f(t))\}$$

By definition of $\mathbf{F}^m$,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} (\overline{\bigotimes}_{f \in \mathbf{F}} \{f(t)\})\}$$

Since all $\{f(t)\}$ are singletons, then $\{f(t)\} \otimes^m \{g(t)\} = \{f(t) \otimes g(t)\}$. Then,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} \{\bigotimes_{f \in \mathbf{F}} f(t)\}\}$$

By definition of C,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\bigcup_{t \in \mathbf{D_X}} \{C(t)\}\}$$

By definition of the set union,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = Sorted^m\{\{C(t) \mid t \in D_{\mathbf{X}}\}\}$$

By definition of the set of ordered m-best elements,

$$sort_{\mathbf{X}}{}^m\{\overline{\bigotimes}_{f \in \mathbf{F}^m} f\} = \{C(t_1), \ldots, C(t_m)\}$$

$\square$

C Proof of Theorem 7

Let $C^m = \{C(t_1), \ldots, C(t_m)\}$ be the m-best solutions of P. Let $\tilde{P}$ be the relaxed version of P solved by *mbe-m-opt*, and let $\tilde{C}^m = \{\tilde{C}(t'_1), \ldots, \tilde{C}(t'_m)\}$ be its m-best solutions. We have to prove that (i) $\tilde{C}^m$ is an m-best upper bound of C^m; and (ii) mbe-m-opt(P) computes $\tilde{C}^m$.

i. It is clear that $\tilde{C}^m = Sorted^m\{C^m \cup W\}$, where W is the set of solutions for which duplicated variables are assigned different domain values. Therefore, by definition, $\tilde{C}^m$ is an m-best bound of C^m.
ii. As shown in Theorem 5, elim-m-opt($\tilde{P}$) computes $\tilde{C}^m$, and by definition of mini-bucket elimination, elim-m-opt($\tilde{P}$) = mbe-m-opt(P). Therefore, mbe-m-opt(P) computes $\tilde{C}^m$.

□

D Proof of Theorem 8

Given a control parameter z, each mini-bucket contains at most z variables. Let deg_i be the number of functions in the bucket $\mathbf{B}_i$ of variable X_i, i.e., the degree of the node in the original bucket tree. Let l_i be the number of mini-buckets created from $\mathbf{B}_i$ and let mini-bucket Q_{i_j} contain deg_{i_j} functions, where $\sum_{j=1}^{l_i} deg_{i_j} = deg_i$. The time compexity of computing a message between two mini-buckets is bounded by $O(k^z m \cdot deg_{i_j} \log m)$ (Proposition 2) and the complexity of computing all messages in mini-buckets created out of $\mathbf{B}_i$ is $O(\sum_{j=1}^{l_i} k^z m \cdot deg_{i_j} \log m) = O(k^z m \cdot deg_i \log m)$. Taking into account that $\sum_{i=1}^{n} deg_i \leq 2n$, we obtain the total runtime complexity of *mbe-m-opt* of $\sum_{i=1}^{n} k^z m \cdot deg_i \log m) = O(nmk^z \log m)$. □

Acknowledgement

This work was supported by NSF grant IIS-1065618.

References

1. S.M. Aji and R.J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.
2. H. Aljazzar and S. Leue. K : A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175:21292154, 2011.
3. S. Bistarelli, H. Faxgier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. Semiring-based CSPs and valued CSPs: Basic properties and comparison. *Over-Constrained Systems*, pages 111–150, 1996.
4. R. I. Brafman, E. Pilotto, F. Rossi, D. Salvagnin, K. B. Venable, and T. Walsh. The next best solution. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA*, 2011.
5. A.W. Brander and M.C. Sinclair. A comparative study of k-shortest path algorithms. In *Proceedings 11th UK Performance Engineering Workshop for Computer and Telecommunications Systems*, pages 370–379, 1995.

6. A. Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.
7. 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.
8. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.
9. R. Dechter and N. Flerova. Heuristic search for m best solutions with applications to graphical models. In *11th Workshop on Preferences and Soft Constraints*, page 46, 2011.
10. R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007.
11. R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM (JACM)*, 50(2):107–153, 2003.
12. P.H. Elliott. Extracting the K Best Solutions from a Valued And-Or Acyclic Graph. Master's thesis, Massachusetts Institute of Technology, 2007.
13. D. Eppstein. Finding the k shortest paths. In *Proceedings 35th Symposium on the Foundations of Computer Science*, pages 154–165. IEEE Comput. Soc. Press, 1994.
14. M. Fromer and A. Globerson. An LP View of the M-best MAP problem. *Advances in Neural Information Processing Systems*, 22:567–575, 2009.
15. H.W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4(1):123–143, 1985.
16. K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying cluster-tree decompositions for automated reasoning. *Artificial Intelligence Journal*, 2005.
17. J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.
18. 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.
19. D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173, 1998.
20. J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.
21. B. Seroussi and J.L. Golmard. An algorithm directly finding the K most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning*, 11(3):205–233, 1994.
22. G. R. Shafer and P.P. Shenoy. Probability propagation. *Anals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.
23. C. Yanover and Y. Weiss. Finding the M Most Probable Configurations Using Loopy Belief Propagation. In *Advances in Neural Information Processing Systems 16*. The MIT Press, 2004.