M best solutions over Graphical Models

Natalia Flerova and Rina Dechter

Department of Computer and Information Science, University of California, Irvine, CA 92697-3425, USA

{nflerova,dechter}@ics.uci.edu

Abstract. Bucket elimination is an algorithmic framework that generalizes dynamic programming to accommodate many problem-solving and reasoning tasks. In particular, it can be used for any combinatorial optimization task such as finding most probable configurations in a Bayesian network. In this paper we present a new algorithm *elim-m-opt*, extending bucket elimination for the task of finding m best solution for an optimization task for any value of m. We formulate our algorithm using general notion of *combination* and *marginalization* operators and show that our approach is sound. We provide complexity analysis and compare it with related work. Potential extension to the mini-bucket framework and its impact on heuristic-search for m-best are discussed.

1 Introduction

The problem of finding the best solution to an optimization problem was thoroughly studied. However, sometimes finding a single best solution is not enough. In many applications it is desirable to obtain a set of several best solutions. In some cases, such as in procurement auction problems, certain constraints of the problem might be too vague or too complicated to formalize and incorporate in the problem definition. It is more efficient to first find several solutions to a relaxed problem and then pick the one that satisfies all the additional constraints. It is sometime desirable to compute several solutions in order to find out how sensitive the optimal one is to a variation of the parameters of the problem. Such tasks arise in biological sequence alignment, for example. In the area of highly reliable communication network design or in genetic linkage analysis (haplotyping), often a set of solutions that have approximately the same cost, but the assignments for which are diverse, is needed.

In this paper we show how the bucket-elimination framework can be extended to compute the m-best solutions by a relatively simple modification of its underlying *combination* and *marginalization* operators [3]. In optimization tasks over probabilistic graphical models the combination operator is a product while the marginalization operator is maximization or minimization (we would assume maximization in this paper). We will show that extending the bucket-elimination algorithm for the m-best solutions can be facilitated by representing functions as vector functions, by defining the combination operator to be a product between vector functions and the marginalization operator as *m-sorting* (rather than maximization), finding the m-highest numbers in a list of function costs and return them as a sorted list. Applying these modifications yields the bucket-elimination algorithm *elim-m-opt*. The actual m-best solution assignments can be assembled by maintaining not just a single best assignment while propagating messages, but rather the m best-cost assignments.

We show that the worst-case complexity of our algorithm increases by a factor of $O(m \cdot \log(m \cdot deg))$ only over finding a single best solution, where deg is the highest degree of the bucket-tree that underlies the computation. This yields an overall complexity of $O(m \cdot n \cdot k^{w^*} \cdot \log(m \cdot deg))$ when k bounds the domain size of each variable and w^* bounds the induced-width/tree-width of the graphical model.

Comparing the algorithm with earlier m-best algorithms we acknowledge upfront that algorithms having better worst-case bounds are already available [14]. Moreover, the algorithm is not the first of its kind; similar algorithms that build upon inference and problem decomposition in graphical models were developed already, and some even long ago. For example, the work by [15] shows how the m-best solutions can be computed over a join-tree. The complexity of their scheme is somewhat higher, $O(m^2 \cdot deg \cdot nk^{w^*})$. Other, more recent,

work [4] extends the m-best solution over SD-DNNF, arriving at comparable complexity bounds. In the next section we will elaborate on these and other related work for the m-best task.

The main virtue of our work however is that it facilitates extensions that carry significant potential. Expressing the m-best task using the general combination and marginalization operators [10] makes the scheme immediately applicable to any inference scheme over a tree-decomposition. In particular, it extends the max-product algorithm over any join-graph for finding the m-best solution. However the most significant aspect of *elim-m-opt* is that it can be applied within the mini-bucket framework, thus facilitating the first known approach for generating heuristic evaluation functions not only for finding the best solution, but also for finding the second best, and third best etc. We will briefly discuss the potential of such heuristics to guide search schemes for finding the m-best solutions towards the end.

Section 2 presents previous work on the m best solution problem. In Section 3 we provide background and preliminaries on graphical models and the bucket-elimination scheme. The contribution of the paper is included in Section 4. In Subsection 4.1 we derive the algorithm ideas using two examples, followed by a general description in Subsection 4.2, where we define the general marginalization and combination. In Subsection 4.3 we address the computational issues of the new combination and marginalization operators and in Subsection 4.4 we give complexity analysis and contrast our algorithm with relevant earlier methods. Section 5 discusses possible future work and applications of the algorithm for heuristic generation for search.

2 Related work

The task of finding m best solutions is closely related to the problem of k shortest paths. A good survey of different k shortest path algorithms can be found in [1] and [5]. Our focus, however, is on the problem of finding m optimal solutions for graphical models. All algorithms discussed in this section are exact, unless specified otherwise.

One of the most influential works on finding the m best solutions was that of Lawler [12]. He improved on the work of Murty [13], who presented an algorithm for ranking of all possible assignments to an optimization problem, and developed a general method that can be applied to a large variety of combinatorial problems. 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 all m best solutions are found. Lawler's scheme has a time complexity linear in number of variables in the problem n and number of solutions generated m, being $O(nm \cdot C(n))$, where $C(n)$ is the complexity of finding a single best solution.

Hamacher and Queyranne built upon Lawler's work [9] and presented a method which assumes the ability to directly find both best and second best solutions to a problem. After finding first 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 $O(m \cdot B(n))$, where $B(n)$ is the complexity of finding the second best solution to the problem. The authors claim that their complexity is always bounded from above by that of Lawler. The method was later used by Hamacher for finding K best network flows [8].

Lawler's approach was applied by Nilsson to a joint-tree [14]. Unlike Lawler or Hamacher and Queyranne, who are solving n problems from scratch in each of iteration of the algorithm, Nilsson is able to utilize the results of previous computations for solving newly formulated problems, using the max-flow propagation. The worst case complexity of the algorithm is $O(m \cdot C(n))$, where $C(n)$ is the complexity of finding a single solution by the max-flow algorithm, which is superior over Lawler's by a factor of n. Yanover and Weiss extended Nilsson's idea for the max-product Belief Propagation algorithm, yielding an approximation algorithm for finding the m best solution for graphs with cycles [19]. Gonzales et al. applied Nilsson's idea for the domain of preference aggregation using graphical utility models [7]. No theoretical bounds on complexity are provided for the two latter algorithms.

Seroussi and Golmard [15] proposed an algorithm that extracts all m solutions from a junction tree directly, i.e., unlike most previously mentioned works, they don't find solutions iteratively from 1^{st} down to m^{th}, but visit in a bottom-up way each clique and at each level store the m best configurations for the previous levels (similar to what our algorithm does, as we will show). Given a junction tree with p cliques, each having at most deg children and the largest clique having k^{w^*} assignments, 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.

Recent work on the subject, which is highly relevant to ours, is the one by Elliot [4], who explores the representation of Valued And-Or Acyclic Graph, i.e., smooth deterministic decomposable negation normal form (sd-DNNF) [2]. The complexity of his algorithm is $O(nk^{w^*} m \log m \cdot deg)$, which for sufficiently large values of m is superior to the complexity of the algorithm by Seroussi and Golmard, but is worse than Nilsson's.

More recently, Fromer and Globerson [6] used a very different approach from the ones mentioned above. They formulated the task of finding m assignments with maximum probability in a probabilistic graphical model as linear program on a particular polytope and solved the problem via a sequence of LP relaxations, yielding an approximation scheme for non-tree graphs.

3 Background and preliminaries

In this section we provide some preliminary definitions. We consider problems expressed as graphical models.

Definition 1 (graphical model, combination, marginalization). *A* graphical model *is a 3-tuple* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$*, where:* $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ *is a set of variables;* $\boldsymbol{D} = \{D_1, \ldots, D_n\}$ *is the set of their finite domains of values;* $\boldsymbol{F} = \{f_1, \ldots, f_r\}$ *is a set of real-valued functions, where each* f_i *is defined over a scope* $S_i \subseteq X$*. The (generalized)* combination operator $\otimes_j f_j$ *is defined over* $U = \cup_j S_j$ *by enumeration as* $\otimes_{j=1}^k f_j \in \{\prod_{j=1}^k f_j, \sum_{j=1}^k f_j, \bowtie_j f_j\}$*. The* marginalization operator $\Downarrow_Y f$ *is defined by enumeration as* $\Downarrow_Y h = \{\max_{S-Y} h, \min_{S-Y} h, \prod_{S-Y} h, \sum_{S-Y} h\}$.

The graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F} \rangle$ represents in a compact form function $F(\mathbf{X}) = \otimes_{i=1}^k f_i$, where $f_i \in \mathbf{F}$.

Definition 2 (reasoning task). *Given a graphical model* $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$*, combination* $\otimes$ *and marginalization* $\Downarrow_Y$*, the* reasoning task *is to compute*

$$f(\boldsymbol{X}) = \Downarrow_{\boldsymbol{X}} \otimes_{j=1}^k f_j \tag{1}$$

Definition 3 (primal graph). *The* primal graph *of a graphical model is an undirected graph,* $G = (X, E)$*, that has variables as its vertices and an edge connects any two variables that appear in the scope of the same function.*

Definition 4 (induced graph, induced width, treewidth). *An* ordered graph *is a pair* (G, d)*, where* G *is an undirected graph, and* $d = (X_1, ..., X_n)$ *is an ordering of the nodes. The* width of a node *in an ordered graph is the number of neighbors that precede it in the ordering. The* width of an ordering d*, denoted* $w(d)$*, is the maximum width over all nodes. The* induced width of an ordered graph*,* $w^*(d)$*, is the width of the induced ordered graph obtained as follows: for each node, from last to first in* d*, its preceding neighbors are connected in a clique. The* induced width of a graph*,* w^**, is the minimal induced width over all orderings. The induced width is also equal to the* treewidth *of a graph.*

An example of a graphical model is shown in Figure 1. The variables are $\{X, Y, T, Z\}$. There are three functions, $f_1(z, t), f_2(x, z), f_3(x, y)$, whose primal graph is depicted. The induced width $w^* = 1$.

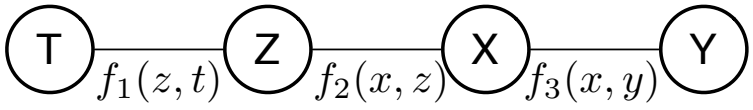


Fig. 1: Problem which bucket-tree is a chain

Definition 5 (bucket elimination). *Bucket elimination (**BE**) is an algorithmic framework that generalizes dynamic programming to accommodate many problem-solving and reasoning tasks [3]. The input to a bucket-elimination algorithm is a graphical model $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$, $F = \{f_1, \cdots, f_k\}$, the combination $\otimes$ and marginalization $\Downarrow_X$ operators, and an ordering $d = (X_1, X_2, \ldots, X_n)$. The ordering d dictates an elimination order for **BE**, from last to first. Each function from **F** is placed in the bucket of its latest variable in d. The algorithm processes the buckets from X_n to X_1, computing for each bucket X_i $\Downarrow_{X_i} \otimes_{j=1}^{n} \lambda_j$, where λ_j are the function in the $bucket_{X_i}$, some are f_i's and some are earlier computed messages. For example, algorithm* elim-opt*, which solves the optimization task, is obtained when $\Downarrow_X f = \max_{S-X} f$ and $\otimes_j = \prod_j$ [3]. 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 d.*

The bucket elimination algorithm is presented in Algorithm 1. The message passing between buckets follow a bucket-tree structure defined next.

Definition 6 (bucket tree). *Bucket elimination constructs a* bucket tree*, by linking the bucket of each X_i to the destination bucket of its message (called the parent bucket). A node of the bucket is associated with its bucket variable.*

Theorem 1 *[3] Given a graphical model and a reasoning task $\mathcal{M} = \langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F}, \otimes, \Downarrow \rangle$ and an ordering d, the* time *and* space complexity *of the bucket elimination algorithm is exponential in the induced width of the ordering.*

.

Algorithm 1 Bucket elimination

Input: A set of functions $F = \{f_1, \cdots, f_n\}$ over scopes $S_1, \cdots, S_n$; An ordering of variables $d = X_1, \cdots, X_n$; A subset Y
Output: A new compiled set of functions from which $\Downarrow_Y \otimes_{i=1}^{n} f_j$ can be derived in linear time
1. **Initialize:** Generate an ordered partition of functions in $bucket_1, \cdots, bucket_n$, where $bucket_i$ contains all the functions whose highest variable in their scope is X_i. Let $S_1, \cdots, S_j$ be the subset of variables in the processed bucket on which functions (new or old) are defined. We assume that any evidence E is absorbed, assigning their values to the functions, and $X = X - E$.
2. **Backward:**
for $p \leftarrow n$ down to 1 **do**
  for all the functions $\lambda_1, \lambda_2, \cdots, \lambda_j$ in $bucket_p$ **do**
    $U_p \leftarrow \cup_{i=1}^{j} S_i \{X_p\}$
    Generate $\lambda_p = \Downarrow_{U_p} \otimes_{i=1}^{j} \lambda_i$
    Place λ_p to the bucket of the largest-index variable in U_p
  end for
end for
3. **Return:** all the functions in each bucket

4 Algorithm *elim-m-opt*

Throughout this paper we will assume without loss of generality that the optimization task at hand is that of maximization, that we use multiplication to combine the functions and that the functions are in fact probability distributions or what sometimes is called "potentials". However, the same algorithm can easily be applied to optimization tasks that use summation or relational join to combine the functions or look for the minimum solutions instead of maximum ones. We will derive the algorithm for finding the m best solutions using two simple examples which will then be followed by the general algorithm.

4.1 Deriving the algorithm using examples

Example 1: consider the problem where a bucket-tree is a chain as shown in Figure 1. Finding the m best solutions to a given problem means finding the m highest joint probability assignments to the function $P(T, Z, X, Y)$, therefore computing

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

where operator $\underset{s \in S}{sort^m} f(s)$ returns the first m elements of the sorted list $\{f(s_i)_{s_i \in D(S)}\}$, and $\underset{y}{sort^m} f(y,x) = \langle f(x,y^1(x)), \ldots, f(x,y^m(x)) \rangle$ is a sorted list of the m best values of $f(x,y)$ for a fixed x. We define the operator $\underset{y}{argsort^m} f(x,y)$ so that $\overline{y(x)} = \underset{y}{argsort^m} f(x,y) = \langle y^1(x), \ldots, y^m(x) \rangle$, where $y^j(x)$ is the j^{th} largest assignment to Y given x relative to $f(x,y)$. It is easy to see that,

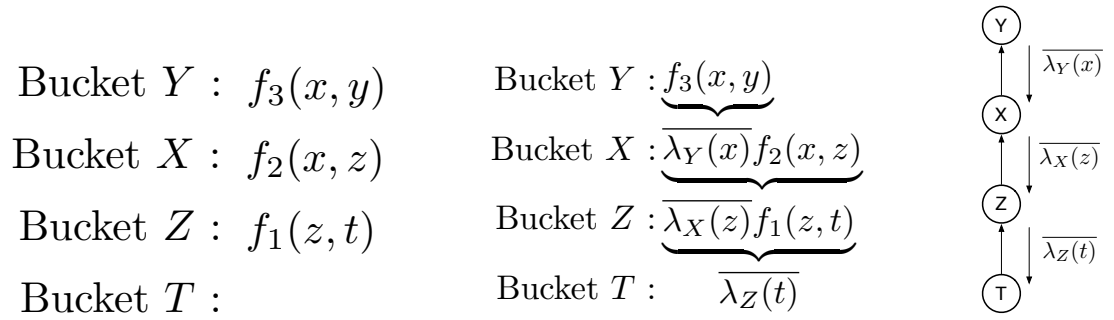
Proposition 1. *Operator sort^m is distributive relative to multiplication.*

Proof. Directly follows from operator $sort^m$ being an extension of the max operator

Hence, as usual, we can apply some 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 get:

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

Distributing of the $sort^m$ operator and carrying computation from right to left of the computed expression is equivalent to organizing the functions into *buckets* along the ordering $d_1 = T, Z, X, Y$ and executing the bucket elimination algorithm *elim-m-opt* for this example, (see Figure 2. First, the functions are parti-



(a) Partition of functions into buckets

(b) Messages between buckets

(c) Bucket tree

Fig. 2: Bucket data structure and messages

tioned into the buckets according to order d_1 (Figure 2a). Next, we process the buckets from last to first,

implementing the right to left computation of Equation 3. $Bucket_Y$ containing a single function $f_3(x,y)$ is processed first. The algorithm applies operator $\underset{y}{sort^m}$ to $f_3(x,y)$ generating a function (also called a message) denoted by $\overline{\lambda_Y(x)}$, which associates each x with the the m best cost vector of values of $f_3(x,y)$:

$$\overline{\lambda_Y(x)} = \underset{y}{sort^m} f_3(x,y) = (\lambda_Y^1(x), \ldots, \lambda_Y^j(x), \ldots, \lambda_Y^m(x)) \tag{4}$$

where $\lambda_Y^j(x)$ is the j^{th} best value of $f_3(x,y)$ given $X = x$. The message $\overline{\lambda_Y(x)}$ is placed in $bucket_X$.

Processing next the bucket of X we compute $\underset{x}{sort^m} f_2(z,x) \cdot \overline{\lambda_Y(x)}$, as dictated by Equation 3, which results in a new message denoted $\overline{\lambda_X(z)}$ to be placed in the $bucket_Z$. Note that $f(x,z) \cdot \overline{\lambda_Y(x)}$ is a multiplication of scalar by a vector and it is placed in $bucket_Z$. Since our bucket-tree is a chain, there will be at most one vector-function $\overline{\lambda}$ in each bucket. We process $bucket_Z$ in a similar manner, generating again an m-cost vector function denoted $\overline{\lambda_Z(t)}$, placed in the $bucket_T$. Finally, processing the last bucket yields a vector of m best solution costs for the entire problem: $Sol = \overline{\lambda_T} = \underset{t}{sort^m} \overline{\lambda_Z(t)}$. The message passing is shown in Figure 2c.

The set of assignments denoted by $\overline{a}$, corresponding to the m best solution costs, can also be generated by propagating the actual variable assignments along with the messages toward to root of the bucket-tree, as follows. When processing $bucket_Y$ we can also generate a vector of the m assignments to Y that correspond to the m largest assignments for each x: $\overline{a_Y}(x) = \underset{y}{argsort^m} f_3(x,y)$ which is placed in $bucket_X$. Subsequently processing $bucket_X$, we will generate the assignments to X as a function of z and concatenate them appropriately with the generated assignments to Y. Namely, $\overline{x}(z) = \underset{x}{argsort^m} f_2(z,x) \cdot \overline{\lambda_Y(x)}$, yielding an extended assignment $\overline{a_X}(z) = \langle \overline{a_Y}(\overline{x}(z)), \overline{x}(z) \rangle$. This vector is placed in $bucket_Z$. Computation continues in a similar manner, until in the last bucket we generate the final vector $\overline{a_T}$ containing the m best assignments to the entire problem.

Example 2: consider now an example where the bucket-tree is not a chain, but a tree, as in Figure 3a. By distributing the $sort^m$ operator relative to the functions in this example, we get:

$$Sol = \underset{t,z,x,y}{sort^m} f_1(t,z) \cdot f_2(z,x) \cdot f_3(z,y) = \underset{t}{sort^m} \underset{z}{sort^m} f_1(t,z) \cdot \underset{x}{sort^m} f_2(z,x) \cdot \underset{y}{sort^m} f_3(z,y) \tag{5}$$

The bucket partition is given in Figure 3b. Buckets $bucket_X$ and $bucket_Y$ are processed as before, generating messages $\overline{\lambda_X(z)}$ and $\overline{\lambda_Y(z)}$, both placed in $bucket_Z$ (see Figure 3b). According to Equation 5 we now need to compute $\underset{z}{sort^m} f_1(t,z) \cdot \overline{\lambda_X(z)} \cdot \overline{\lambda_Y(z)}$ which expresses the m best solutions below Z relatively to the bucket tree in Figure 3a. Therefore we should combine the scalar function $f_1(t,z)$ with the two messages $\lambda_X(z)$ and $\lambda_Y(z)$, each containing the m best costs of solutions of subproblems below X and Y as a function of z, respectively. The resulting combination is a new vector function that has the m^2 best solution costs for each z, each equals to the product of two elements coming from the different messages. This combination operation, which we denote by $\overline{\otimes}$, (to be defined below) is a Cartesian product of the m-vector functions $\overline{\lambda_X(z)}$ and $\overline{\lambda_Y(z)}$. We then apply $sort^m$ to the resulting combination, selecting the m best solutions out of those m^2. The resulting message $\overline{\lambda_Z(t)}$ correspond to:

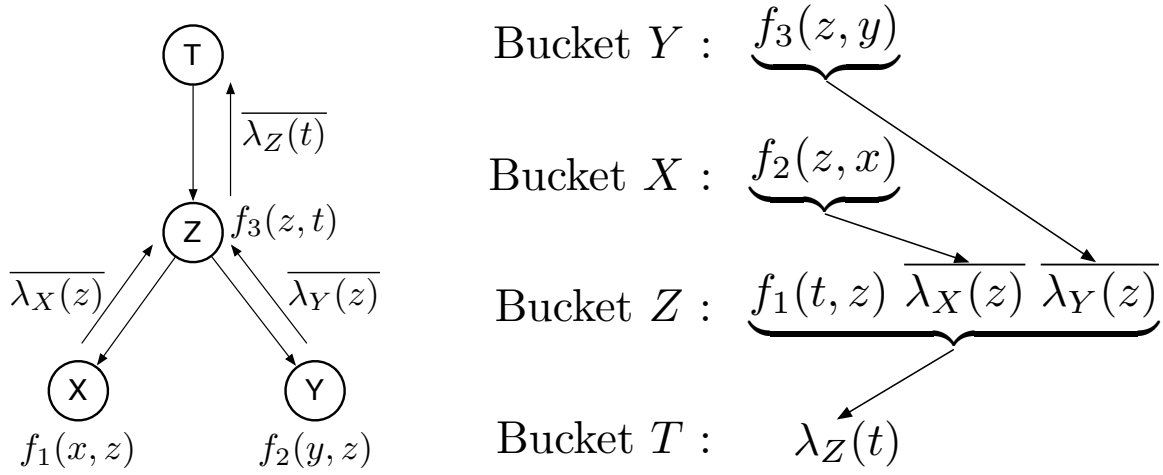
$$\overline{\lambda_Z(t)} = \underset{z}{sort^m} f_1(t,z) \cdot \overline{\lambda_X(z)} \otimes \overline{\lambda_Y(z)} \tag{6}$$

The rest of the computation is the same as in example 1.

4.2 The general algorithm

To describe the general algorithm we need the following notations and definitions.

Definition 7. *A **vector function** is a function from its scalar domain to a real valued vector. The dimension of the function is the length of the vector. The **product of two vector functions**, $f_1(S_1)$ having scope S_1 and dimension m, and $f_2(S_2)$ having scope S_2 and dimension n, is a function $f_3(S_3)$ of dimension $m \cdot n$ over the scope $S_3 = S_1 \cup S_2$ such that for $t \in D_{S_3}$, $f_3(t) = \langle f^{i,j}(t)|_{i=1..m,j=1..n} \rangle$ and $f^{i,j}(t) = f_1^i(t_{S_1}) \cdot f_2^j(t_{S_2})$.*



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

Fig. 3: Problem with a bucket-tree which is not a chain

Definition 8. ***Vector function combination.*** *Given a set of vector functions $\{f_1(S_1), \cdots, f_n(S_n)\}$, the* ***vector combination*** *operator $\overline{\otimes_{i=1}^n} f_i(S_i)$ is the multiplication of the vector functions.*

Definition 9. *Given a set of real numbers $S = \{s_1, \cdots, s_l\}$, $l \geq m$, we define $sort^m(S)$ as $sort^m(S) = (s^{(1)}, \cdots, s^{(m)})$ where $s^{(i)} \in S$ and $s^{(1)} \geq s^{(2)} \geq, \cdots, \geq s^{(m)}$ and for all others $s \in S$ $s \leq s^{(m)}$*

Definition 10. *Given a vector function of dimension m over scope S and $X \in S$, we define the sort* ***marginalization*** *operator $\Downarrow_X^m = \underset{X}{sort^m} f$ having scope S as follows*

$$\underset{X}{sort^m} f(t) = \underset{X}{sort^m} f(t,x) = sort^m\{f(t,x_1), f(t,x_2), \cdots, f(t,x_k)\}. \tag{7}$$

where $x_1, \cdots, x_k$ are the k values in the domain of X.

Definition 11 (m-best solution). *The* m-best solution task *is defined over a graphical model $\langle X, D, F\rangle$, whose combination function is the vector multiplication and whose marginalization operator is m-sort*

The bucket-elimination algorithm *elim-m-opt* described in Figure 2 uses the two new combination and marginalization operators of $\overline{\otimes}$ and $sort^m$. The algorithm processes the buckets from last to first as usual. Each bucket contains some vector functions and some scalar functions, and in each bucket an output vector function is generated and placed in the appropriate bucket. The function generated in each bucket is described in step 2 of the algorithm and its computation is detailed in the next subsection.

The correctness of the algorithm can be deduced by showing that the combination and marginalization defined for m-best algorithm obey axioms that guarantee the correctness of the algorithm as part of a general class as described in [16, 17, 11].

Proposition 2. *Given a graphical model, the operators of combination $\overline{\otimes}$ and marginalization $\underset{X}{sort^m}$ over vector functions satisfy the following properties (first formulated in general in [16],[17]):*

1. *Order of marginalization does not matter: $\underset{X-\{X_i\}}{sort^m}(\underset{X-\{X_j\}}{sort^m} f(X)) = \underset{X-\{X_j\}}{sort^m}(\underset{X-\{X_i\}}{sort^m} f(X))$*
2. *Commutativity: $g\overline{\otimes}f = f\overline{\otimes}g$*
3. *Associativity: $f\overline{\otimes}(g\overline{\otimes}h) = (f\overline{\otimes}g)\overline{\otimes}h$*
4. *Restricted distributivity: $\underset{X-\{X_k\}}{sort^m}[f(X-\{X_k\})\overline{\otimes}g(X)] = f(X-\{X_k\})\overline{\otimes}\underset{X-\{X_k\}}{sort^m} g(X)$*

Proof. This can be proved by relaying on the fact that multiplication and maximization, on which our extended combination is defined, obey the above 4 axioms.

Theorem 2 *Algorithm* elim-m-opt *is sound and complete for finding the m-best solutions over a graphical model.*

Proof. Follows from previous work on cluster-tree decomposition, of which a bucket tree is a special case [11] or, alternatively, from the works of Shanoy [16],[17] . They showed that when combination and marginalization satisfy the above 4 axioms, algorithms such as bucket elimination (and some more general ones) are sound and complete.

Algorithm 2 *elim-m-opt* algorithm

Input: A set of functions $F = \{f_1, \cdots, f_n\}$ over scopes $S_1, \cdots, S_n$; An ordering of variables $d = X_1, \cdots, X_n$; A subset Y, and $\otimes$ and $\Downarrow$ operators of product and maximization
Output: An m-vector function equal to $\Downarrow_Y \otimes_{i=1}^n f_j = \underset{x_1,\cdots,x_n}{sort^m} \otimes_{i=1}^n f_j$, where $\otimes$ is a vector combination operator; The m largest assignments to variables $\{X_1, \cdots, X_n\}$, $\overline{a} = \underset{x_1,\cdots,x_n}{argsort} \otimes_{i=1}^n f_j$

1. **Initialize:** Generate an ordered partition of functions in $bucket_1, \cdots, bucket_n$, where $bucket_i$ contains all the functions whose highest variable in their scope is X_i. Let $S_1, \cdots, S_j$ be the subset of variables in the processed bucket on which functions (new or old) are defined. We assume that any evidence E is absorbed into functions and $X = X - E$.
2. **Backward:**
for $p \leftarrow n$ down to 1 **do**
  for all the functions $\lambda_1, \lambda_2, \cdots, \lambda_j$ in $bucket_p$ **do**
    $U_p \leftarrow \cup_{i=1}^j S_i\{X_p\}$
    Generate $\overline{\lambda_p(U_i)} = \Downarrow_{U_p} \overline{\otimes}_{i=1}^j \lambda_i$ using *process-bucket* $(\lambda_1, ..., \lambda_j)$.
    Generate $\overline{a_p(U_i)} = \underset{X_i}{argsort^m} \overline{\otimes}_{i=1}^j \lambda_i$
    Place $\lambda_p(U_i)$ and $\overline{a_p}(U_i)$ to the bucket of the largest-index variable in U_p
  end for
end for
3. **Return:** function $\overline{\lambda_{X_1}}$ and assignment $\overline{a_{X_1}}$ generated in the first bucket in the ordering.

4.3 Processing a bucket

We will next discuss the computation of an output function by procedure *process-bucket*. Let us go back to our example in Figure 3a and to the the output function in bucket z, namely to compute, for every $T = t$

$$\overline{\lambda_Z(t)} = \underset{Z}{sort^m} f_3(z,t) \overline{\otimes} \overline{\lambda_Y(z)} \overline{\otimes} \overline{\lambda_X(z)} \tag{8}$$

Assuming bi-valued variables having domains $\{0, 1\}$, the task is to select the m best elements from two lists containing m^2 pair-wise products each (we assume the algorithm is described for a fixed t):

$$L_{Z=0} = \langle f_3(Z=0,t) \cdot \lambda_Y^i(Z=0) \cdot \lambda_X^j(Z=0) \rangle_{i,j \in (1..m)} \tag{9}$$

$$L_{Z=1} = \langle f_3(Z=1,t) \cdot \lambda_Y^i(Z=1) \cdot \lambda_X^j(Z=1) \rangle_{i,j \in (1..m)} \tag{10}$$

Instead of generating the m^2 elements in the product and then sorting, we will exploit the fact that each of the two lists $\overline{\lambda_Y(z)}$ and $\overline{\lambda_X(z)}$ are themselves sorted. Since $\lambda_Y^1(0) \geq \lambda_Y^2(0) \geq \cdots \geq \lambda_Y^m(0)$ and

$\lambda_X^1(0) \geq \lambda_X^2(0) \geq \cdots \geq \lambda_X^m(0)$, the largest element in the output list $L_{Z=0}$ can be generated by multiplying $\lambda_X^1(0)$ and $\lambda_Y^1(0)$. We denote this product element by $e_{\langle 1,1\rangle}(0)$ and its cost is $V_{<1,1>}(0) = \lambda_X^1(0) \cdot \lambda_Y^1(0) \cdot f_3(0)$. The second best element of $L_{Z=0}$ can be obtained by replacing one of the costs in the above product by a second-best from either $\lambda_X(0)$ or $\lambda_Y(0)$, but not both. Namely, once $e_{\langle 1,1\rangle}(t)$ is selected , we can consider the best next from its "child" elements $e_{\langle 1,2\rangle}(z)$ or $e_{\langle 2,1\rangle}(z)$ having costs denoted $V_{<1,2>(z)}$ and $V_{<1,1>(z)}$, where:

$$V_{<1,2>}(0) = \lambda_X^1(0) \cdot \lambda_Y^2(0) \cdot f_3(0), \tag{11}$$

$$V_{<2,1>}(0) = \lambda_X^2(0) \cdot \lambda_Y^1(0) \cdot f_3(0) \tag{12}$$

The same applies for $Z = 1$, yielding $V_{<1,1>}(1)$, as best solution and $V_{<1,1>}(1)$ or $V_{<2,1>}(1)$ as second best of $L_{Z=1}$. The process of selecting the m best solutions from the union of the lists $L_{Z=0}$ and $L_{Z=1}$ can therefore described as a greedy search from a set of disconnected trees of candidate elements or nodes $e_{\langle i,j\rangle}(z)$ associated with a value $V_{<i,j>}(z) = \lambda_X^i(z) \cdot \lambda_Y^j(z) \cdot f_3(z)$.

Thus, a two-dimensional element $e_{\langle i,j\rangle}$ has 2 children $e_{\langle i+1,j\rangle}$ and $e_{\langle i,j+1\rangle}$ where the value of a child is always less or equal to its parent. This suggests an algorithm, in which we grow two disconnected trees. At each step, the best value node among all the leaves (of the 2 trees) is selected to be the next best solution, its children are generated and appended as leaves, and the process iterates. Since we need just m best solutions the number of steps will be exactly m. The selection of the best among all leaves can be facilitated by maintaining the leaves in a sorted OPEN list. The generated search trees for $m = 4$ in our example is giving in Figure 4. In general a bucket of a variable X contains vector functions (and scalar functions, which are vector functions of dimension one) $\overline{\lambda_1(S_1)}, \cdots, \overline{\lambda_k(S_k)}$ over scopes $S_1, \cdots, S_k$ and we want to generate the m-dimensional function $\overline{\lambda_X(S)}$, $S = \cup_i S_i - X$. For every $t \in D_S$ we grow a set of k disconnected trees of candidate best solutions, each for a different value of X. The roots of each tree is $e_{\langle 1,1,\cdots,1\rangle}(x)$. The child of each node is generated by incrementing a single dimension and its cost is computed appropriately. At each step, the algorithm selects the largest among all the leaves as its next best solution, generates its child nodes, computes their values and then insert them in the right places in OPEN. The algorithm for a fixed assignment t in the domain of the generated function is described in Algorithm 3.

Algorithm 3 Bucket processing

Input: $bucket_X$ of variable X containing a set of m-vector functions $\{\overline{\lambda_1(S_1, X)}, \cdots, \overline{\lambda_d(S_d, X)}\}$, assignment $t \in D_S$, where $S = \cup_{i=1}^d S_i - X$.
Output: m-vector function $\overline{\lambda_X(S)}$.
$n_0(x) \leftarrow \langle 1, 1, \cdots, 1 \mid X = x\rangle$; {create roots for each tree of $X = x$}
OPEN order $\leftarrow \{n_0(x_1), ..., n_0(x_k)\}$ {put the best in each list in descending order of $V_{<i_1,\cdots,i_d>}(x)$};
$j = 0$; {initialize the numbered solution index}
while $j \leq m$, by $+1$ **do**
  $n \leftarrow$ first element $n = \langle i_1, \cdots, i_d\rangle(x)$ in OPEN. Remove n from OPEN;
  $\lambda_X^j(s) \leftarrow n$; {the j^{th} element is selected}
  $C \leftarrow$ children(n) $= \{\langle i_1, \cdots, i_r + 1, \cdots, i_d\rangle(x)\}$;
  Place each $c \in C$ in its right place in OPEN based on its computed value. Check for duplicates;
end while

4.4 Complexity

The complexity of processing a bucket dictates the overall complexity of the algorithm.

Proposition 3. *Given a bucket of a variable X over scope S having j functions $\{\lambda_1, ..., \lambda_j\}$ of dimension m, where m is the number of best-solutions sought and k bounds the domain size, the complexity of processing the bucket is $O(k^{|S|} \cdot m \cdot j \log(m \cdot j))$, where $|S|$ is the scope size of S.*

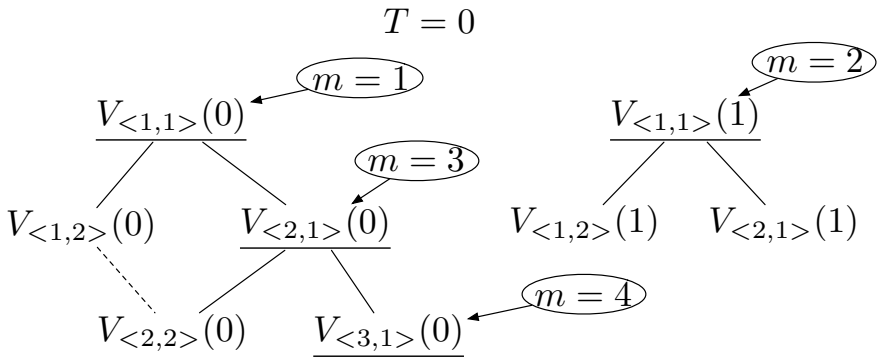


Fig. 4: Search space

Proof. We apply the algorithm to each tuple in the domain of $S - X$ which amounts to $k^{|S|}$ tuples. For each, algorithm *process-bucket* has m steps and in each the next best element is selected from a sorted list OPEN, then its j possible child nodes are generated and inserted into OPEN based on their values. Inserting j elements to a sorted list whose length is at most $m \cdot j$ takes $O(m \cdot j \cdot log(m \cdot j))$. Therefore the total complexity of processing the bucket is $O(k^{|S|}(m \cdot j) \cdot log(m \cdot j))$

Since there are n buckets, one for each variable X_i, each having scope S_i whose size is bounded by the induced width w^*, and given that the number of functions in $bucket_i$ is deg_i, the total time complexity of *elim-m-opt* is $T = \sum_{i=1}^{n} O(k^{w^*} m \cdot deg_i \log(m \cdot deg_i))$. Assuming $deg_i \leq deg$ we get:

$$T = O(k^{w^*} m \log(m \cdot deg) \sum_{i=1}^{n} deg_i) = O(k^{w^*} m \cdot \log(m \cdot deg) \cdot 2n) = O(nmk^{w^*} \log(m \cdot deg)) \qquad (13)$$

The space complexity equals to the total number of candidate solutions we need to keep at the same time, $O(k^{w^*} m \cdot deg_i)$, plus the size of the partial assignments to the variable that we need to propagate which is $O(mnk^{w^*})$. So overall space complexity is dominated by $O(mnk^{w^*})$. In summary:

Theorem 3 *Given a graphical model $\langle \boldsymbol{X}, \boldsymbol{D}, \boldsymbol{F} \rangle$ having n variables, an ordering d, induced-width of w^*, a bucket-tree degree bounded by deg and the domain size bounded by k, the time complexity of finding the m best solution by* elim-m-opt *algorithm is $O(k^{w^*} nm \cdot \log(m \cdot deg))$ and its space complexity is $O(mnk^{w^*})$.*

A quick comparison with earlier related work shows that when specialized to a bucket tree the best complexity bound among the works surveyed is that of Nilsson's algorithm [14], which is dominated by $O(nmk^{w^*})$. The complexity of Elliot's algorithm [4] can be bounded by $O(nk^{w^*} m \log(m \cdot deg))$, the same as *elim-m-opt*. The algorithm by Seroussi and Golmard [15] has the complexity $O(nk^{w^*} m^2 deg)$, which is inferior to *elim-m-opt* or Elliot by a factor of $\frac{m \cdot deg}{\log(m \cdot deg)}$. The complexity of Lawler's scheme [12], assuming we use *elim-opt* to find the best solution, is $O(m \cdot n^2 k^{w^*})$. For $n < m \cdot deg$ it is worse than all other algorithms. Figure 5 depicts the dominance relationships between the algorithms in terms of time complexity. A parent node in the graph has a better complexity than its children.

5 Future work: on the potential use of elim-m-opt

The complexity of *elim-m-opt*, as all bucket elimination algorithms and the three of algorithms mentioned above, is exponential in the tree-width w^*, which limits the potential of direct use of the *elim-m-opt* for solving m best task. However, *elim-m-opt* can have an important extension to the mini-bucket elimination, yielding the approximation algorithm *mb-elim-m-opt*. Mini-bucket scheme approximates the computations

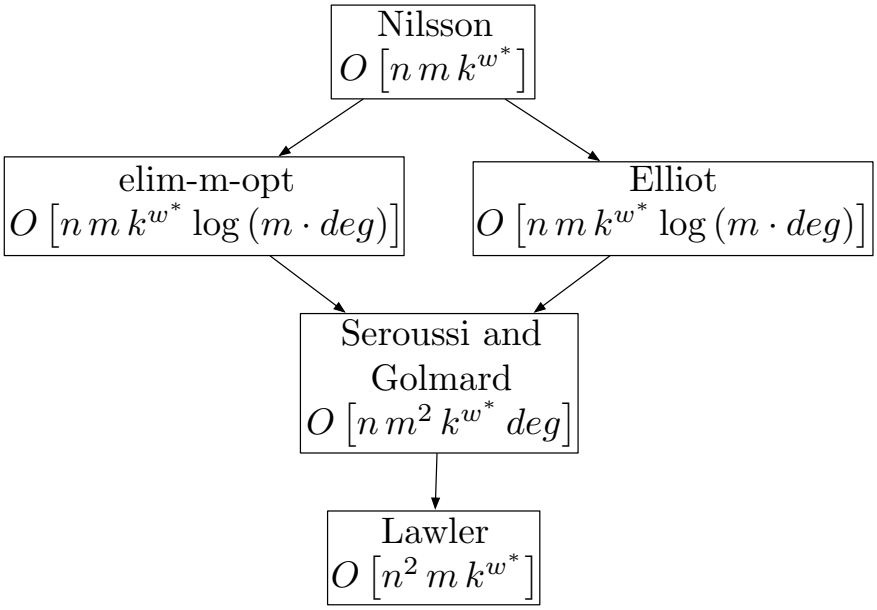


Fig. 5: Complexity comparison

by partitioning buckets into mini-buckets, and then eliminating the appropriate variables in each mini-bucket independently, providing upper bounds on the best solution cost [10].

The primary use of mini-bucket elimination is to generate heuristics that can guide both best first and depth first search. Both search algorithms can be extended to the task of finding m best solutions. In the case of Best First search it is possible to find m best solutions in a non-increasing order of their costs by simply continuing search after finding the first solution. This idea was indeed used in [18] where best-first search was applied to Weighted Boolean Function Directed Acyclic Graph for finding m best solutions for the MAP problem.

Depth-first Branch and Bound can be extended to find the m best solution when pruning only occurs when the heuristic upper bound on the subproblem is worst than the m^{th} lower bound we have (remember we look for a maximization).

Algorithm *mb-elim-m-opt* can generate a set of m upper bounds on the m best solution values for the subproblem below each node, yielding a set of m heuristic estimates. These m heuristics can be used in Best First search to decide the need to re-open explored nodes. In Branch and Bound the availability of the m heuristic estimates will allow fine-tune pruning. For example, assume that for a node N we have lower bounds on solution costs $\{L_1(N), L_2(N), \cdots, L_m(N)\}$ of $C_1, ..., C_m$, denoting the exact values of the m-best solutions. When only a single heuristic evaluation function of the node $f(N)$ is available, we can prune the subproblem below N if $f(N) \leq L_m(N)$ only. However, if we have m heuristics $f_1(N), \cdots, f_m(N)$ of the m best costs below N, we can prune below N if $f_1(N) \leq L_1(N)$ and $f_2(N) \leq L_2(N)$ and $\cdots$ and $f_m(N) \leq L_m(N)$. This pruning condition is stronger and will therefore yields smaller search space. The details of extending of Best First and Branch and Bound algorithms to the m best solution problems remain future work.

Acknowledgements

This work was supported by NSF grant IIS-1065618.

References

1. AW 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.
2. A. Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.
3. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85, 1999.
4. P.H. Elliott. Extracting the K Best Solutions from a Valued And-Or Acyclic Graph. Master's thesis, Massachusetts Institute of Technology, 2007.
5. D. Eppstein. Finding the k shortest paths. In *Proceedings 35th Symposium on the Foundations of Computer Science*. IEEE Comput. Soc. Press, 1994.
6. M. Fromer and A. Globerson. An LP View of the M-best MAP problem. *Advances in Neural Information Processing Systems*, 22:567–575, 2009.
7. C. Gonzales, P. Perny, and S. Queiroz. Preference aggregation with graphical utility models. In *Proceedings AAAI-08*, pages 1037–1042, 2008.
8. H.W. Hamacher. A note on K best network flows. *Annals of Operations Research*, 57(1):65–72, 1995.
9. H.W. Hamacher and M. Queyranne. K best solutions to combinatorial optimization problems. *Annals of Operations Research*, 4(1):123–143, 1985.
10. K. Kask, J. Larrosa, and R. Dechter. Up and down mini-buckets: A scheme for approximating combinatorial optimization tasks. Technical report, University of California Irvine, 2001.
11. Kalev Kask, Rina Dechter, Javier Larrosa, and Fabio Cozman. Bucket-tree elimination for automated reasoning. *Artif. Intel*, 125:91–131, 2001.
12. 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.
13. K.G. Murty. An algorithm for ranking all the assignments in increasing order of cost. *Operations Research*, 16:682–687, 1968.
14. D. Nilsson. An efficient algorithm for finding the M most probable configurations in probabilistic expert systems. *Statistics and Computing*, 8(2):159–173, 1998.
15. B. Seroussi and JL Golmard. An algorithm directly finding the K most probable configurations in Bayesian networks. *International Journal of Approximate Reasoning*, 11(3):205–233, 1994.
16. P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. *Uncertainty in Artificial Intelligence*, 4:169–198, 1990.
17. P.P. Shenoy. Binary join trees for computing marginals in the Shenoy-Shafer architecture. *International Journal of Approximate Reasoning*, 17(2-3):239–263, 1997.
18. S.E. Shimony and E. Charniak. A new algorithm for finding MAP assignments to belief networks. In *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pages 185–196. Elsevier Science Inc., 1990.
19. 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.