On Mini-Buckets and the Min-fill Elimination Ordering

Emma Rollon and Javier Larrosa

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Spain.

Abstract. *Mini-Bucket Elimination* (MBE) is a well-known approximation of *Bucket Elimination* (BE), deriving bounds on quantities of interest over *graphical models*. Both algorithms are based on the sequential transformation of the original problem by eliminating variables, one at a time. The order in which variables are eliminated is usually computed using the greedy min-fill heuristic. In the BE case, this heuristic has a clear intuition, because it faithfully represents the structure of the sequence of sub-problems that BE generates and orders the variables using a greedy criteria based on such structure. However, MBE produces a sequence of sub-problems with a different structure. Therefore, using the min-fill heuristic with MBE means that decisions are made using the structure of the sub-problems that BE would produce, which is clearly meaningless. In this paper we propose a modification of the min-fill ordering heuristic that takes into account this fact. Our experiments on a number of benchmarks over two important tasks (i.e., computing the probability of evidence and optimization) show that MBE using the new ordering is often far more accurate than using the standard one.

1 Introduction

The graphical model paradigm includes very important reasoning tasks such as solving and counting solutions of CSPs, finding optimal solutions of weighted CSPs, computing probability of evidences and finding the most probable explanation in Bayesian Networks. Mini-Bucket Elimination (MBE) [5] is a very popular algorithm deriving bounds on reasoning tasks over graphical models. The good performance of MBE in different contexts has been widely proved [5, 7, 11, 12].

MBE is a relaxation of Bucket Elimination (BE) [2] and both algorithms work by eliminating the problem variables, one at a time. In BE, the order in which variables are eliminated is important because it determines the complexity of the algorithm. In MBE, the variable elimination order does not affect the complexity of the algorithm, which comes determined by a control parameter. However, as we show in this paper, such order greatly affects the accuracy of the bound.

The most common elimination order for both BE and MBE is the one given by the min-fill greedy heuristic [3]. This heuristic was originally designed for BE and it is there where it has a clear rationale: the greedy algorithm that computes the min-fill ordering takes into account the structure of the sequence of sub-problems that BE will subsequently produce. Thus, each time the algorithm decides the next variable to be eliminated it does so by considering the structure of the problem that BE will have at this point.

Using the same heuristic for MBE does not seem a good idea, because MBE produces a sequence of subproblems with a different structure. Thus, when MBE uses the min-fill heuristic it takes decisions based on a misleading information.

In this paper we show that a better elimination ordering for MBE may be computed by considering the real structure of its sequence of subproblems. To do that, we represent these subproblems by *induced z-bounded hyper-graphs* and compute the elimination ordering accordingly. We demonstrate that MBE using the new elimination ordering is often far more accurate than using the standard one on a number of benchmarks (i.e., *coding networks*, real-world *genetic linkage analysis*, real-world *noisy-OR* models, and *combinatorial auctions*) over two tasks (i.e., computing the probability of evidence, and finding the complete assignment with minimum cost in a WCSP).

2 Background

2.1 Graphical Models

A *graphical model* is a tuple $(\mathcal{X}, \mathcal{F})$, where $\mathcal{X} = (x_1, \dots, x_n)$ is an ordered set of variables and $\mathcal{F} = \{f_1, \dots, f_r\}$ is a set of functions. Variable x_i takes values from its finite domain $\mathcal{D}_i$. Each function $f_j : \mathcal{D}_{var(f_j)} \rightarrow A$ is defined over a subset of variables $var(f_j) \subseteq \mathcal{X}$ and returns values from a set A. For example, $\mathcal{X} = (x_1, x_2)$ with $\mathcal{D}_1 = \mathcal{D}_2 = \{0, 1\}$, and $\mathcal{F} = \{x_1 + x_2, x_1 * x_2\}$ is a graphical model. Abusing notation, the scope of a set of functions $\mathcal{F}$, noted $var(\mathcal{F})$, is the union of scopes of the functions it contains.

Given a graphical model, one can compute different *reasoning tasks*. A reasoning task is defined by two operations ($\bigotimes$ and $\Downarrow$) over functions. The *combination* of f and g, noted $f \bigotimes g$, is a new function h with scope $var(h) = var(f) \cup var(g)$, while the marginalization of a set of variables $\mathcal{W} \subseteq \mathcal{X}$ from function f, noted $f \Downarrow_{\mathcal{W}}$, is a new function h with scope $var(h) = var(f) - \mathcal{W}$. Computing the reasoning task means computing $(\bigotimes_{f \in \mathcal{F}} f) \Downarrow_{\mathcal{X}}$

The graphical model framework can be used to model a variety of important combinatorial problems. For example, if $\mathcal{F}$ is a set of cost functions (i.e, returning a non-negative value representing a cost) the graphical model is a weighted CSP. If we take the sum as combination and the minimum as marginalization, the reasoning task becomes $\min_{\mathcal{X}}\{\sum_{f \in \mathcal{F}} f)\}$, which is the minimum cost assignment of the weighted CSP. Alternatively, if $\mathcal{F}$ is a set of *conditional probability tables* we have a Bayesian Network. If we take the product as combination and the sum as marginalization, the reasoning task becomes $\sum_{\mathcal{X}}\{\prod_{f \in \mathcal{F}} f)\}$, which models the probability of the evidence. If $\mathcal{F}$ is a set of hard constraints (i.e, boolean functions) the graphical model is a classical CSP and the reasoning task $\sum_{\mathcal{X}}\{\prod_{f \in \mathcal{F}} f)\}$ counts its solutions.

2.2 Graph concepts

The structure of a graphical model is represented by its associated *hyper-graph*.

Definition 1. *A hyper-graph H is a pair $H = (V, E)$ where V is a set of elements, called* nodes*, and E is a set of non-empty subsets of V, called* hyper-edges*. The width of hyper-graph H is the size of its largest edge.*

Definition 2. *Given a graphical model $P = (\mathcal{X}, \mathcal{F})$, its associated hyper-graph $H(P) = (V, E)$ is defined as $V = \{i \mid x_i \in \mathcal{X}\}$ and $E = \{var(f) \mid f \in \mathcal{F}\}$.*

The most fundamental structural property considered in the context of graphical models is *acyclicity*. Mainly, acyclicity is measured in terms of the *induced width*.

Definition 3. *Let $H = (V, E)$ be a hyper-graph, and let $o = \{x_1^o, \ldots, x_n^o\}$ be an ordering of the nodes in V where x_j^o is the j^{th} element in the ordering. This induces a sequence of hyper-graphs $H_n, H_{n-1}, \ldots, H_1$ where $H = H_n$ and H_{j-1} is obtained from H_j as follows. All edges in H_j containing x_j^o are merged into one edge, called the induced hyper-edge, and then x_j^o is removed. Thus, the underlying vertices of H_{j-1} are $x_1^o, \ldots, x_{j-1}^o$. The induced width of H under o, noted $w^*(o)$, is the largest width among all hyper-graphs $H_n, \ldots, H_1$. The induced width of H, noted w^*, is the minimum induced width over all orderings o.*

Example 1. Consider a graphical model $P = (\mathcal{X}, \mathcal{F})$ where $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ and $\mathcal{F} = \{f_1(x_1, x_3), f_2(x_2, x_3), f_3(x_2, x_4), f_4(x_1, x_4)\}$. Its hyper-graph is $H(P) = (V, E)$, where $V = \{1, 2, 3, 4\}$ and $E = \{(1, 3), (2, 3), (2, 4), (1, 4)\}$. The lexicographical ordering $o = \{x_1, x_2, x_3, x_4\}$ induces the following sequence of hyper-graphs (where each hyper-graph is represented by its set of hyper-edges):

$$
\begin{aligned}
&H_4(P) = \{(1, 3), (2, 3), (2, 4), (1, 4)\} \\
&H_3(P) = \{(1, 3), (2, 3), (1, 2)\} \\
&H_2(P) = \{(1, 2)\} \\
&H_1(P) = \{(1)\}
\end{aligned}
$$

The induced width of the problem is 2 - all edges in $H_4(P)$, $H_3(P)$ and $H_2(P)$ achieve this size.

2.3 Bucket Elimination

Bucket Elimination (BE) [2] (*non-serial dynamic programming* in [1] and *fusion algorithm* in [13]) is a general algorithm for the computation of reasoning tasks in graphical models. BE (Algorithm 1) works as a sequential elimination of variables. Given an arbitrary variable ordering $o = \{x_1^o, \ldots, x_n^o\}$ (line 1), the algorithm eliminates variables one by one, from last to first, according to o. The elimination of variable x_j^o is done as follows: $\mathcal{F}$ is the set of current functions. The algorithm computes the so called *bucket* of x_j^o, noted $\mathcal{B}_j$, which contains all cost functions in $\mathcal{F}$ having x_j^o in their scope (line 3). Next, BE computes a new function g_j by combining all functions in $\mathcal{B}_j$ and subsequently eliminating x_j^o (line 4). Then, $\mathcal{F}$ is updated by removing the functions in $\mathcal{B}_j$ and adding g_j (line 5). The new $\mathcal{F}$ does not contain x_j^o (all functions mentioning x_j^o were removed) but preserves the value of the result. The elimination of the last variable produces an empty-scope function (i.e., a constant) which is the result of the problem (line 7).

The correctness of the algorithm is guaranteed whenever the combination and marginalization operators satisfy the three *Shenoy-Shaffer axioms* [13]. The most important tasks over graphical models satisfy these axioms.

Algorithm 1: Bucket Elimination

Input : A graphical model $P = (\mathcal{X}, \mathcal{F})$.
Output: Evaluation of $(\bigotimes_{f \in \mathcal{F}} f) \Downarrow_{\mathcal{X}}$.

$\{x_1^o, \ldots, x_n^o\} \leftarrow$ `compute-order(`P`);`
for $j \leftarrow n$ **to** *1* **do**
  $\mathcal{B}_j \leftarrow \{f \in \mathcal{F} | \; x_j^o \in var(f)\}$;
  $g_j \leftarrow (\bigotimes_{f \in \mathcal{B}_j} f) \Downarrow_{x_j^o}$;
  $\mathcal{F} \leftarrow (\mathcal{F} \cup \{g_j\}) - \mathcal{B}_j$;
end
return $\bigotimes_{f \in \mathcal{F}} f$;

Algorithm 2: Mini-Bucket Elimination

Input : A graphical model $P = (\mathcal{X}, \mathcal{F})$; and the value of the control parameter z.
Output: A bound of $(\bigotimes_{f \in \mathcal{F}} f) \Downarrow_{\mathcal{X}}$.

$\{x_1^o, \ldots, x_n^o\} \leftarrow$ `compute-order(`P`);`
for $j \leftarrow n$ **to** *1* **do**
  $\mathcal{B}_j \leftarrow \{f \in \mathcal{F} | \; x_j^o \in var(f)\}$;
  $\{Q_1, \ldots, Q_p\} \leftarrow$ `partition(`$\mathcal{B}_j$, z `);`
  for $k \leftarrow 1$ **to** p **do**
    $g_{j,k} \leftarrow (\bigotimes_{f \in Q_k} f) \Downarrow_{x_j^o}$;
  end
  $\mathcal{F} \leftarrow (\mathcal{F} \cup \{g_{j,1}, \ldots, g_{j,p}\}) - \mathcal{B}_j$;
end
return $\bigotimes_{f \in \mathcal{F}} f$;

Example 2. Consider the graphical model in Example 1. The trace of BE along lexicographical order is as follows.

Bucket	
$\mathcal{B}_4$	$f_4(x_1, x_4)$, $f_3(x_2, x_4)$
$\mathcal{B}_3$	$f_1(x_1, x_3)$, $f_2(x_2, x_3)$, $g_4(x_1, x_2) = (f_4 \bigotimes f_3) \Downarrow_{x_4}$
$\mathcal{B}_2$	$g_3(x_1, x_2) = (f_1(x_1, x_3) \bigotimes f_2(x_2, x_3) \bigotimes g_4(x_1, x_2)) \Downarrow_{x_3}$
$\mathcal{B}_1$	$g_2(x_1) = g_3(x_1, x_2) \Downarrow_{x_2}$
Output	$g_1() = g_2(x_1) \Downarrow_{x_1}$

Since new functions have to be stored explicitly as tables, and their size is exponential on their arity, the time and space complexity of BE depends on the largest arity needed. This arity is captured by the structural parameter induced-width (see Section 3 for details).

Theorem 1. *Given a variable ordering o, the time and space complexity of BE is $O(exp(w^*(o) + 1))$ and $O(exp(w^*(o)))$, respectively.*

2.4 Mini-Bucket Elimination

All variable elimination algorithms are unsuitable for problems with high induced width due to its exponential time and space complexity. *Mini-bucket elimination* (MBE) [5] is an approximation of full bucket elimination that bounds the exact solution when the induced width is too large.

Given a bucket $\mathcal{B}_j = \{f_1 \ldots, f_m\}$, MBE generates a partition $Q = \{Q_1, \ldots, Q_p\}$ of $\mathcal{B}_j$, where each subset $Q_k \in Q$ is called *mini-bucket*. Given an integer control parameter z, MBE restricts the arity of each of its mini-buckets to $z+1$. We say that Q is a *z-partition*. Then, each mini-bucket is processed independently. Algorithm 2 shows the pseudo-code of MBE.

Example 3. Consider our running example. The trace of MBE along lexicographical order and setting the value of the control parameter z to 1 is as follows.

Bucket	
$\mathcal{B}_4$	$f_4(x_1, x_4)$, $f_3(x_2, x_4)$
$\mathcal{B}_3$	$f_1(x_1, x_3)$, $f_2(x_2, x_3)$
$\mathcal{B}_2$	$g_{42}(x_2) = f_3(x_2, x_4) \Downarrow_{x_4}$, $g_{32}(x_2) = f_2(x_2, x_3) \Downarrow_{x_3}$
$\mathcal{B}_1$	$g_{41}(x_1) = f_4(x_1, x_4) \Downarrow_{x_4}$, $g_{31}(x_1) = f_1(x_1, x_3) \Downarrow_{x_3}$
Output	$g_1() = (g_{41}(x_1) \bigotimes g_{31}(x_1)) \Downarrow_{x_1}$, $g_2() = (g_{42}(x_2) \bigotimes g_{32}(x_2)) \Downarrow_{x_2}$

Note that since the final set of functions is $\{g_1(), g_2()\}$, the output valuation is $g_1() \bigotimes g_2()$.

The time and space complexity of MBE is $O(exp(z+1))$ and $O(exp(z))$, respectively. The parameter z allows trading time and space for accuracy. In general, higher values of z results in more accurate bounds. In the limit (e.g., when z is the number of variables of the problem) MBE behaves as BE and computes the exact result.

3 Variable Elimination Ordering

In this Section we show that the order in which variables are eliminated plays a very different role in bucket and mini-bucket elimination. In particular, the sequence of subproblems generated by both algorithms is different. In spite of this key distinction, MBE uses the ordering procedure as designed for BE. We propose a modification of this procedure in order to account for this fact.

3.1 Induced hyper-graphs and BE

There exists a close relation between the induced sequence of hyper-graphs and the elimination process of BE. The trace of BE in Example 2 showed how it is possible to compute the scopes of the functions that the algorithm will produce without actually executing it. Since the hyper-graph precisely contains this information, we can easily show that the sequence of induced hyper-graphs is actually the sequence of hyper-graphs associated with the sequence of subproblems produced by BE.

Algorithm 3: compute-order

Input : A graphical model $(\mathcal{X}, \mathcal{F})$, and a variable selection heuristic h.
Output: A variable elimination ordering $\{x_1^o, \ldots, x_n^o\}$.

for $j \leftarrow n$ **to** 1 **do**
  $x_j^o \leftarrow \arg\min_{x_i \in \mathcal{X}} \{h(H(P_j), i)\}$;
end
return $\{x_1^o, \ldots, x_n^o\}$;

Algorithm 4: compute-z-order

Input : A graphical model $(\mathcal{X}, \mathcal{F})$, and a variable selection heuristic h.
Output: A variable elimination ordering $\{x_1^o, \ldots, x_n^o\}$.

for $j \leftarrow n$ **to** 1 **do**
  $x_j^o \leftarrow \arg\min_{x_i \in \mathcal{X}} \{h(H(\overline{P}_j), i)\}$;
end
return $\{x_1^o, \ldots, x_n^o\}$;

Given a graphical model $P = (\mathcal{X}, \mathcal{F})$, let P_{j-1} be the subproblem produced by BE once variables $x_j^o, \ldots, x_n^o$ have been eliminated, where by definition $P_n = P$. P_{j-1} is obtained from P_j by computing a new function g_j with scope $var(\mathcal{B}_j) - \{x_j^o\}$, and removing the variable from the problem. Similarly, by definition $H_n = H(P)$, and induced hyper-graph H_{j-1} is obtained from H_j by merging all hyper-edges containing x_j^o and then removing x_j^o from the set of vertices. Note that the new hyper-edge is the scope of g_j, while the other hyper-edges are the scopes of the remaining functions in P_j. Therefore, H_{j-1} is the associated hyper-graph of P_{j-1} (i.e., $H_{j-1} = H(P_{j-1})$).

Example 4. Consider our running example and its BE trace in Example 2. Subproblems P_j are the following:

$$
\begin{aligned}
&P_4 = \{f_1(x_1, x_3), f_2(x_2, x_3), f_3(x_2, x_4), f_4(x_1, x_4)\} \\
&P_3 = \{f_1(x_1, x_3), f_2(x_2, x_3), g_4(x_1, x_2)\} \\
&P_2 = \{g_4(x_1, x_2), g_3(x_1, x_2)\} \\
&P_1 = \{g_2(x_1)\}
\end{aligned}
$$

Note that the set of functions' scopes in each subproblem P_j corresponds to the edges in hyper-graph $H_j(P)$ in Example 1.

It is clear then that the induced width bounds the bucket's sizes generated during the elimination process and, as a consequence, the complexity of the algorithm. The size of the induced width varies with various variable orderings, leading to different performance guarantees. Finding the *best* ordering (i.e., the one with the smallest induced width) is NP-hard. Instead, useful variable selection heuristics as fill-in edges [3], and width of nodes [6] aim at finding *good* orderings.

Procedure `compute-order` (Algorithm 3) is a greedy search guided by the variable selection heuristic h defined on a hyper-graph $H = (V, E)$ and one node $i \in V$, noted $h(H, i)$. At iteration j, the algorithm selects the j^{th} variable in the ordering (i.e., x_j^o) by ranking each node in subproblem P_j according to h and selecting the one minimizing it. Note that since the induced hyper-graph $H_j(P)$ represents subproblem P_j, the algorithm selects the best variable in the problem once variables $x_{j+1}^o, \ldots, x_n^o$ has been eliminated.

3.2 Induced z-bounded hyper-graphs and MBE

The sequence of induced hyper-graphs differ from the sequence of hyper-graphs associated with subproblems produced by MBE. The reason is that MBE partitions buckets whenever they have more than $z+1$ different variables.

Given a graphical model $P = (\mathcal{X}, \mathcal{F})$, let $\overline{P}_{j-1}$ be the subproblem once MBE has eliminated variables $x^o_j, \dots, x^o_n$ from P, where by definition $\overline{P}_n = P$. Consider that MBE does not partition buckets $\mathcal{B}_j, \dots, \mathcal{B}_n$. Up to this point of the execution, MBE generates the same subproblems as BE (i.e., $P_j = \overline{P}_j, \dots, P_n = \overline{P}_n$) and the induced hyper-graphs correspond to hyper-graphs associated with these subproblems (i.e., $H_j = H(\overline{P}_j), \dots, H_n = H(\overline{P}_n)$). Now consider that bucket $\mathcal{B}_{j-1}$ has more than $z+1$ different variables. MBE will partition this bucket into mini-buckets. Namely, instead of computing a single function g_j over the bucket's scope, the algorithm will compute a set of functions g_{jk} over unions of scopes of bucket's functions. The hyper-graph associated with subproblem $\overline{P}_{j-1}$ would have one hyper-edge for each of the new functions' scope, while the induced hyper-graph H_{j-1} has only one hyper-edge over the scope of the bucket. Therefore, $H_{j-1} \neq H(\overline{P}_{j-1})$.

Example 5. Consider our running example and the trace of MBE in Example 3. Subproblems $\overline{P}_j$ are as follows:

$$
\begin{aligned}
&\overline{P}_4 = \{f_1(x_1,x_3), f_2(x_2,x_3), f_3(x_2,x_4), f_4(x_1,x_4)\}\\
&\overline{P}_3 = \{f_1(x_1,x_3), f_2(x_2,x_3), g_{41}(x_1), g_{42}(x_2)\}\\
&\overline{P}_2 = \{g_{31}(x_1), g_{32}(x_2), g_{41}(x_1), g_{42}(x_2)\}\\
&\overline{P}_1 = \{g_2(), g_{31}(x_1), g_{41}(x_1)\}
\end{aligned}
$$

Note that induced hyper-graphs $H_3(P)$ and $H_2(P)$ in Example 1 are not associated with subproblems $\overline{P}_3$ and $\overline{P}_2$, respectively. The reason is that bucket $\mathcal{B}_4$ is partitioned into mini-buckets $\{f_4(x_1,x_4)\}$ and $\{f_3(x_2,x_4)\}$. The new computed functions are $g_{41}(x_1)$ and $g_{42}(x_2)$. None of the functions in $\overline{P}_3$ has scope $\{x_1,x_2\}$. However, the induced hyper-graph $H_3(P)$ has an hyper-edge on $\{x_1,x_2\}$.

Although this important difference, most previous investigations on MBE uses the elimination ordering as designed for BE. This does not seem a good decision because, as we have seen, the variable selection heuristic h ranks each node according to the given hyper-graph. Therefore, when computing the ordering for MBE, the heuristic selects the next variable to eliminate based on an *erroneous* structure.

We wish to compute the ordering over the hyper-graphs associated with each subproblem generated by MBE. Let us call *z-bounded hyper-graph*, the hyper-graph associated with subproblem $\overline{P}_j$ for any $j = 1 \dots n$, and *induced z-bounded hyper-graphs*, the sequence of hyper-graphs associated with the sequence of subproblems $\overline{P}_1, \dots, \overline{P}_n$.

Example 6. Consider the trace of MBE in Example 3. The sequence of associated induced z-bounded hyper-graphs (represented by their hyper-edges) is,

$$
\begin{aligned}
&H(\overline{P}_4) = \{(1,3),(2,3),(2,4),(1,4)\}\\
&H(\overline{P}_3) = \{(1,3),(2,3),(1),(2)\}\\
&H(\overline{P}_2) = \{(1),(2)\}
\end{aligned}
$$

Iteration j	`compute-order`	`compute-z-order`
4	$H(P_4) = \{(1,3),(2,3),(2,4),(1,4)\}$	$H(\overline{P}_4) = \{(1,3),(2,3),(2,4),(1,4)\}$
	$h(\cdot,4) = 1$	$h(\cdot,4) = 1$
	$h(\cdot,3) = 1$	$h(\cdot,3) = 1$
	$h(\cdot,2) = 1$	$h(\cdot,2) = 1$
	$h(\cdot,1) = 1$	$h(\cdot,1) = 1$
	$x_j^o = 4$	$x_j^o = 4$
3	$H(P_3) = \{(1,3),(2,3),(1,2)\}$	$H(\overline{P}_3) = \{(1,3),(2,3),(1),(2)\}$
	$h(\cdot,3) = 0$	$h(\cdot,3) = 1$
	$h(\cdot,2) = 0$	$h(\cdot,2) = 0$
	$h(\cdot,1) = 0$	$h(\cdot,1) = 0$
	$x_j^o = 3$	$x_j^o = 2$
2	$H(P_2) = \{(1,2)\}$	$H(\overline{P}_2) = \{(1,3),(3),(1)\}$
	$h(\cdot,2) = 0$	$h(\cdot,3) = 0$
	$h(\cdot,1) = 0$	$h(\cdot,1) = 0$
	$x_j^o = 2$	$x_j^o = 3$
1	$H(P_1) = \{(1)\}$	$H(\overline{P}_1) = \{(1)\}$
	$h(\cdot,1) = 0$	$h(\cdot,1) = 0$
	$x_j^o = 1$	$x_j^o = 1$

Fig. 1. Trace of `compute-order` and `compute-z-order` using number of fill-in edges as variable selection heuristic h (ties are broken lexicographically). The value of z is 1.

$$H(\overline{P}_1) = \{(1)\}$$

We propose to compute the elimination order according to the induced z-bounded hyper-graphs. We call this procedure `compute-z-order` (Algorithm 4). The main difference with respect to `compute-order` is that, at iteration j, the variable selection heuristic h will rank nodes in the z-bounded hyper-graph $H(\overline{P}_j)$ instead of ranking nodes in the hyper-graph $H(P_j)$ (line 2 in both algorithms). Note that in the limit (e.g., when z is the number of variables in the problem) both `compute-order` and `compute-z-order` are equivalent.

Example 7. Consider our running example. Let the variable selection heuristic h be number of fill-in edges, and let z be 1. In case of ties, the secondary variable selection heuristic is lexicographical order. Figure 1 shows the behavior of `compute-order` and `compute-z-order`. In summary, procedure `compute-order` outputs order $o = \{x_1, x_2, x_3, x_4\}$ while `compute-z-order` outputs order $o' = \{x_1, x_3, x_2, x_4\}$. Note that under o, MBE will split buckets $\mathcal{B}_4$ and $\mathcal{B}_3$ into two mini-buckets each. However, under o', MBE will split only bucket $\mathcal{B}_4$ into two mini-buckets and compute exactly the remaining buckets. As a consequence, the bound will provably be more accurate using o' (which is based on induced z-bounded hyper-graphs) than using o (which is based on induced hyper-graphs).

Since `compute-z-order` needs subproblems $\overline{P}_j$, computing the order as a preprocess could have the same complexity as MBE. However, it can be embedded in MBE

Algorithm 5: Mini-Bucket Elimination with embedded compute-z-order

Input : A graphical model $P = (\mathcal{X}, \mathcal{F})$; and the value of the control parameter z.
Output: A bound of $(\bigotimes_{f \in \mathcal{F}} f) \Downarrow_{\mathcal{X}}$.

for $j \leftarrow n$ **to** *1* **do**
  $x_j^o \leftarrow \arg\min_{x_i \in \mathcal{X}} \{h(H(\mathcal{X}, \mathcal{F}), i)\}$; `// At each iteration` $\overline{P}_j = (\mathcal{X}, \mathcal{F})$
  $\mathcal{B}_j \leftarrow \{f \in \mathcal{F} | \, x_j^o \in var(f)\}$;
  $\{Q_1, \ldots, Q_p\} \leftarrow$ `partition(`$\mathcal{B}_j$, z `)`;
  for $k \leftarrow 1$ **to** p **do**
    $g_{j,k} \leftarrow (\bigotimes_{f \in Q_k} f) \Downarrow_{x_j^o}$;
  end
  $\mathcal{F} \leftarrow (\mathcal{F} \cup \{g_{j,1}, \ldots, g_{j,p}\}) - \mathcal{B}_j$;
  $\mathcal{X} \leftarrow \mathcal{X} - \{x_j^o\}$;
end
return $\bigotimes_{f \in \mathcal{F}} f$;

(Algorithm 5). Note that the time and space complexity of the new algorithm remains exponential on the control parameter z.

4 Empirical Evaluation

The good performance of mini-bucket elimination over different reasoning tasks has been already proved [5, 7, 11, 12] . The purpose of these experiments is to evaluate the effectiveness of the new min-fill heuristic adapted to MBE over two important tasks: (i) computing the probability of evidence over Bayesian networks, and (ii) finding the minimum cost assignment of the weighted CSP.

We conduct our empirical evaluation on four benchmarks: coding networks, real-world linkage analysis models, real-world noisy-OR networks, and combinatorial auctions. The task on the first three benchmarks (all of them included in the UAI'08 evaluation [1]) is to compute the probability of evidence and MBE obtains upper bounds, while the task on the latter benchmark is optimization and MBE obtains lower bounds.

When computing the probability of evidence, we report the results using two different bucket partitioning policies as described in [12]: scope-based (SCP) and LMRE content-based heuristic. We use the number of fill-in edges as variable selection heuristic h with `compute-order` and `compute-z-order` (in the following called *BE fill-in* and *MBE fill-in*, respectively).

Unless otherwise indicated, we report the results in tables where the first column identifies the instance. Then, for each bucket partitioning heuristic we report the bound, relative error (RE), and cpu time in seconds using *BE fill-in* and *MBE fill-in*. For each instance, the relative error is computed as

$$RE = \frac{|bound - best\ bound|}{best\ bound}$$

[1] http://graphmod.ics.uci.edu/uai08/Software

BN inst.'s number	SCP partition heuristic						LMRE partition heuristic					
	BE fill-in			MBE fill-in			BE fill-in			MBE fill-in		
	ub.	RE	Time	ub.	RE	Time	ub.	RE	Time	ub.	RE	Time
$z = 20$												
126	1.31E-44	46.96	5.44	**2.72E-46**	0	6.7	2.49E-45	8.15	26.78	**1.15E-45**	3.23	24.87
127	**1.10E-49**	0	7.44	2.75E-46	2491.16	7.87	**1.73E-46**	1568.50	33.03	2.03E-46	1836.54	35.86
128	1.37E-41	127.18	7.17	**1.28E-42**	11.00	7.38	5.87E-41	547.40	34.54	**1.07E-43**	0	31.48
129	1.77E-46	333.65	6.2	**5.46E-47**	101.99	6.5	2.41E-44	45574.38	25.36	**5.30E-49**	0	27.84
130	8.22E-47	535.66	6.54	**1.53E-49**	0	6.45	1.03E-47	66.30	23.7	**8.44E-48**	54.11	24.59
131	5.03E-46	547.72	6.9	**9.16E-49**	0	5.41	2.28E-46	248.36	27.64	**1.96E-47**	20.36	25.94
132	1.29E-46	43.84	6.89	**1.04E-47**	2.61	6.44	1.05E-47	2.65	29.34	**2.88E-48**	0	24.66
133	**5.03E-46**	1.85	6.61	2.97E-45	15.82	7.28	2.70E-42	15296.10	24.93	**1.76E-46**	0	28.79
134	2.50E-44	8513.76	6.66	**2.94E-48**	0	6.69	**4.06E-45**	1381.71	29.69	5.49E-45	1866.70	29.45
$z = 22$												
126	5.21E-43	3.98E+5	26.14	**5.72E-46**	437.12	25.15	9.70E-45	7422.72	107.16	**1.31E-48**	0	101.11
127	5.34E-48	0.68	28.17	**3.18E-48**	0	28.22	**2.26E-47**	6.11	108.68	2.76E-45	865.35	125.69
128	2.30E-44	1.23	25.58	**1.03E-44**	0	25.02	9.03E-42	872.98	130.68	**1.96E-43**	17.98	114.71
129	6.14E-45	5.19E+4	26.85	**1.18E-49**	0	26.04	3.65E-43	3.08E+6	89.18	**3.35E-47**	282.32	90.08
130	8.40E-47	1205.90	21.64	**1.61E-49**	1.31	24.1	2.49E-48	34.73	90.47	**6.96E-50**	0	77.76
131	**9.86E-48**	0.25	21.69	6.09E-47	6.72	24.59	2.71E-46	33.32	93.29	**7.88E-48**	0	81.37
132	1.46E-48	23.50	23.6	**5.96E-50**	0	20.52	1.49E-48	24.03	93.17	**5.44E-49**	8.13	90.88
133	8.50E-44	1327.32	23.05	**8.66E-45**	134.26	24.68	5.21E-45	80.37	99.4	**6.40E-47**	0	85.04
134	1.57E-46	5.32	26.5	**1.09E-46**	3.36	28.92	1.01E-46	3.07	105.31	**2.49E-47**	0	94.95

Table 1. Empirical results on coding networks. BN_126, …, BN_134 instances.

Moreover, for each row we underline the best bound, and highlight in bold face the best bound wrt each bucket partitioning heuristic.

In all our experiments, we execute MBE in a Pentium IV running Linux with 4 Gb of memory and 3 GHz.

Coding networks. Our first domain is coding networks from the class of linear block codes [7]. All instances have 512 variables with domain size 2 and the induced width varies from 49 to 55. Table 1 shows the results for two different values of the control parameter $z = \{20, 22\}$.

The MBE fill-in computes the best upper bound on eight out of nine instances when $z = 20$, and on all instances when $z = 22$. Among these instances, the improvement over the best BE fill-in is usually of orders of magnitude for both values of z.

Using the SCP partitioning heuristic, the MBE fill-in outperforms the BE fill-in on seven instances when $z = 20$ and on eight instances when $z = 22$. The improvements are usually of orders of magnitude. The computation times of both orderings are very close. Using the LMRE partitioning heuristic, the MBE fill-in outperforms the BE fill-in on seven instances when $z = 20$, and on eight instances when $z = 22$. As for the previous partitioning heuristic, the improvements are in general of orders of magnitude, and the computation times are similar.

For space reasons, we do not report the number of mini-buckets processed in each run of MBE. However, we observed that when using MBE fill-in the algorithm processes less mini-buckets than when using the BE fill-in. Note that breaking a bucket into several mini-buckets is precisely what transforms the variable elimination scheme from exact (i.e, BE) to approximate (i.e., MBE). The less mini-buckets, the more similar to the exact algorithm and, as a consequence, the more accurate the bound.

pedigree instance's number	SCP partition heuristic						LMRE partition heuristic					
	BE fill-in			MBE fill-in			BE fill-in			MBE fill-in		
	ub.	RE	Time	ub.	RE	Time	ub.	RE	Time	ub.	RE	Time
$z = 17$												
7	1.34E-49	1.22E+4	4.29	**2.22E-51**	202.01	4.92	1.84E-53	0.68	8.54	**1.10E-53**	0	18.14
9	2.58E-66	136.76	1.76	**1.88E-68**	0	2.51	1.94E-67	9.32	2.72	**5.39E-68**	1.87	2.71
13	3.24E-15	9.43E+4	1.91	**1.39E-16**	4059.24	2.23	**3.43E-20**	0	2.54	1.42E-16	4129.41	3.00
18	4.15E-71	24.54	0.86	**3.47E-72**	1.13	0.95	2.08E-71	11.80	0.92	**1.63E-72**	0	0.98
20	3.82E-25	4.73	12.83	**6.66E-26**	0	15.57	**7.61E-26**	0.14	14.15	9.75E-26	0.46	23.16
25	1.57E-109	8.83	0.56	**1.99E-110**	0.24	0.65	3.33E-109	19.85	0.68	**1.60E-110**	0	0.65
30	4.57E-75	2418.39	1.46	**1.86E-77**	8.86	1.43	7.14E-76	376.82	1.65	**1.89E-78**	0	1.75
31	9.21E-51	1.17E+5	9.82	**6.67E-53**	849.54	11.5	**7.84E-56**	0	12.08	3.94E-52	5020.51	13.45
33	**1.70E-47**	17.05	3.53	5.33E-45	5645.31	5.3	**9.44E-49**	0	10.84	2.92E-46	308.02	8.50
34	2.97E-49	3.39E+4	32.81	**1.62E-51**	183.85	37.25	3.31E-53	2.79	49.75	**8.74E-54**	0	63.39
37	4.94E-109	7052.31	110.36	**3.19E-111**	44.45	131.21	8.86E-110	1263.00	243.17	**7.01E-113**	0.00	235.84
39	**2.58E-99**	0.09	1.25	7.35E-99	2.12	1.04	**2.35E-99**	0	1.28	6.76E-99	1.87	1.42
41	1.96E-61	19.36	69.22	**1.48E-62**	0.54	29.16	1.06E-61	9.98	90.5	**9.62E-63**	0	487.54
42	**1.22E-26**	0.00	15.84	1.69E-26	0.38	39	**1.50E-26**	0.23	25.7	3.71E-26	2.03	51.06
44	5.81E-55	131.49	1.99	**4.39E-57**	0	3.08	**4.10E-56**	8.35	3.58	8.08E-56	17.43	3.69
51	**1.74E-53**	862.93	3.09	9.76E-52	48484.18	3.53	**2.01E-56**	0	5.12	6.59E-56	2.27	4.43
$z = 19$												
7	**1.35E-53**	814.96	24.52	1.63E-50	981031.51	29.49	6.20E-56	2.74	29.24	**1.65E-56**	0	65.00
9	7.37E-67	9869.66	6.27	**8.57E-70**	10.47	6.43	1.34E-68	177.77	11.01	**7.47E-71**	0	12.15
13	**2.01E-18**	45.03	6.51	1.24E-15	28356.80	8.44	**4.36E-20**	0	9.98	3.58E-17	821.45	12.60
18	**4.16E-76**	0	2.72	5.43E-76	0.31	2.78	2.92E-75	6.03	2.76	**1.58E-75**	2.81	2.81
20	2.24E-27	1.11	51.53	**1.52E-27**	0.43	43.69	1.12E-27	0.06	51.02	**1.05E-27**	0.00	87.88
25	4.87E-111	1.76	1.38	**4.50E-111**	1.55	1.94	**1.77E-111**	0	1.49	4.21E-111	1.38	1.94
30	**5.46E-80**	0	5.8	9.05E-80	0.66	5.29	1.03E-79	0.89	6.12	**8.02E-80**	0.47	5.40
31	**7.04E-56**	15.60	37.3	1.79E-55	41.28	48.23	**4.24E-57**	0	42.78	2.48E-56	4.86	60.05
33	**3.30E-46**	120.34	15.81	7.89E-46	289.04	17.38	**2.72E-48**	0	15.9	1.03E-46	36.70	24.13
34	**1.69E-51**	851.77	181.91	7.32E-51	3690.45	223.95	**1.98E-54**	0	331.9	1.71E-53	7.61	427.58
37	3.75E-113	0.11	206.35	**3.53E-113**	0.04	203.48	**3.39E-113**	0	368.57	5.99E-113	0.77	319.95
39	2.08E-100	2.07	8.85	**6.77E-101**	0	6.75	**1.35E-100**	0.99	8.88	1.89E-100	1.79	6.84
41	**3.00E-63**	309.19	266.96	1.14E-61	11784.67	311.98	1.82E-63	187.18	496.47	**9.68E-66**	0	612.98
42	2.01E-27	1.27	156.51	**1.37E-27**	0.55	203.89	1.14E-27	0.29	184.97	**8.84E-28**	0	198.82
44	6.39E-55	62.59	7.36	**5.63E-55**	55.05	10.34	**1.01E-56**	0	12.37	7.39E-55	72.60	13.57
51	1.10E-55	269.98	12.21	**4.07E-58**	0	13.66	**1.11E-55**	271.12	14.31	2.26E-55	554.91	14.53

Table 2. Empirical results on linkage analysis. Pedigree instances.

Linkage analysis. Our second domain is real-world linkage analysis models. We used pedigree instances. They have 300 to 1000 variables with domain sizes from 1 (i.e., evidence variables) to 5, and induced widths of 20 up to 50. Table 2 shows the results.

The MBE fill-in computes the best upper bound on ten out of sixteen instances when $z = 17$, and on seven instances when $z = 19$. Among these instances, the improvement over the best BE fill-in is of orders of magnitude on eight out of ten (i.e., on 80% of) instances when $z = 17$, and on five out of seven (i.e., on 71%) when $z = 19$. Among instances where the BE fill-in computes the best upper bound, the improvement over the best MBE fill-in is of orders of magnitude on three out of six (i.e., on 50%) instances when $z = 17$, and on five out of nine (i.e., on 55%) when $z = 19$. In other words, when better, the MBE fill-in is usually orders of magnitude more accurate.

Using the SCP partitioning heuristic, the MBE fill-in outperforms the BE fill-in on twelve instances when $z = 17$, and on eight when $z = 19$. Among these instances, the improvement over the BE fill-in is always of orders of magnitude when $z = 17$, and from 6% up to orders of magnitude (on 37% of these instances) when $z = 19$. Using

or_chain numbers	Size	Mean RE				or_chain numbers	Size	Mean RE			
		SCP heuristic		LMRE heuristic				SCP heuristic		LMRE heuristic	
		BE fill-in	MBE fill-in	BE fill-in	MBE fill-in			BE fill-in	MBE fill-in	BE fill-in	MBE fill-in
1[0*]	12	916951	**165.80**	595.83	**9.85**	21*	10	97025.5	**365.01**	298.49	**1.03**
11*	10	163720	**1918.51**	8785.5	**13.05**	22*	10	4.98E+06	**20.77**	493.017	**6.34**
12*	9	343487	**80.44**	25293	**1.94**	23*	9	2.31E+09	**126119**	1.24E+06	**188.58**
13*	9	1.52E+07	**303.44**	10236.7	**5491.76**	24*	9	548805	**113.44**	4001.65	**23.55**
14*	10	54466.7	**135.01**	759.94	**1.79**	25*	4	510.92	**93.02**	**54.37**	93.62
15*	10	199.96	**17.95**	35.92	**9.86**	3*	10	1.00E+10	**3.02E+07**	4673.18	**0.11**
16*	10	8.64E+07	**58875.3**	468.50	**4.43**	4*	10	461584	**33.68**	2301.8	**1.30**
17*	9	1.25E+10	**9375.63**	311467	**12.66**	50*	9	1.64E+06	**1788.8**	71460.5	**7.69**
18*	9	5.50E+07	**204569**	19986.2	**2.13**	6*	10	1.06E+10	**1.15E+07**	2293.18	**10.81**
19*	9	1.17E+06	**2983.57**	208.47	**111.36**	7*	9	1.98E+10	**56410.1**	982912	**8.86**
2[0*]	10	3.59E+06	**126.63**	980.99	**42.66**	8*	9	207240	**4411.55**	90128.1	**19.75**

Table 3. Empirical results on noisy-OR networks. Promedas instances.

the LMRE partitioning heuristic, the MBE fill-in outperforms the BE fill-in on eight instances when $z = 17$, and on seven instances when $z = 19$. Among these instances, the improvement over the BE fill-in is of orders of magnitude on all of them when $z = 17$, while on six out of seven (i.e., on 87.5%) instances when $z = 19$.

Computation times show the same behavior as in the previous benchmark. Regarding the number of mini-buckets, a smaller number of mini-buckets is in general attached to a better accuracy. This suggests a heuristic strategy to select the ordering in a preprocessed way by selecting the order producing the smallest number of mini-buckets (or, equivalently, the smallest number of new induced hyper-edges).

Noisy-OR networks. Our third domain is real-world noisy-OR networks generated by the Promedas expert system for internal medicine [14]. The benchmark contains 238 instances having 23 up to 2133 variables (mean number is 1048) with binary domain sizes and induced width up to 60.

Table 3 summarizes the results for $z = 25$. We report the mean relative error among sets of instances. The first column identifies the instances included in each set as a regular expression. For example, the first row includes instances with names matching or_chain_1[0*] (e.g., or_chain_1, or_chain_10, or_chain_101, etc). The second column indicates the size of the set. As before, we underline the best relative error for each set of instances, and highlight in bold face the best relative error wrt each partition heuristic. We do not report cpu time because its behavior is the same as for the previous benchmarks.

The MBE fill-in outperforms the BE fill-in on all sets, with the exception of or_chain_25* (which only has 4 instances). The improvement among the best BE fill-in is always of orders of magnitude. Using the SCP partitioning heuristic, the MBE fill-in clearly outperforms the BE fill-in on all sets, while when using the LMRE partitioning heuristic, the MBE fill-in is superior to the BE fill-in on all sets but or_chain_25*.

Combinatorial auctions. Our last domain is combinatorial auctions (CA). They allow bidders to bid for indivisible subsets of goods. We have generated CA using the path model of the CATS generator [8]. We experiment on instances with 20 and 50 goods,

nb. bids	z	nb. goods = 20				nb. goods = 50			
		BE fill-in		MBE fill-in		BE fill-in		MBE fill-in	
		lb.	RE	lb.	RE	lb.	RE	lb.	RE
80	15	493	0.010	**498**	0	**424.9**	0	423.3	0.004
85	15	371.5	0.010	**375.3**	0	439.4	0.010	**443.8**	0
90	15	**463.4**	0.000	458.4	0.011	**400.7**	0.000	398.1	0.006
95	15	524.9	0.001	**525.3**	0	458.6	0.016	**466.1**	0
100	15	577.5	0.018	**588.1**	0	498.8	0.004	**500.9**	0
105	15	549.9	0.009	**555.1**	0	**550.1**	0	539.2	0.020
110	15	631.7	0.001	**632.1**	0	587.9	0.008	**592.7**	0
115	15	604	0.023	**618**	0	581.2	0.006	**585**	0
120	15	498.4	0.018	**507.7**	0	555.6	0.016	**564.9**	0
125	15	659.8	0.003	**662**	0	**566.2**	0	558.9	0.013
130	15	623.1	0.017	**633.9**	0	**550.6**	0	534.9	0.029
135	15	734.4	0.009	**740.7**	0	**562.2**	0	560.2	0.004
140	15	765.5	0.015	**776.8**	0	702.9	0.015	**713.9**	0
145	15	746.9	0.001	**748**	0	502.8	0.025	**515.8**	0
150	15	680.7	0.012	**689**	0	**697.5**	0.000	696.5	0.001
155	15	**671.4**	0	666	0.008	647	0.016	**657.6**	0
160	15	744.4	0.002	**745.8**	0	777.3	0.029	**800.9**	0
165	15	808.9	0.013	**819.2**	0	667.3	0.019	**680**	0
170	15	**707.6**	0	**707.6**	0	586.9	0.048	**616.2**	0
175	15	812.7	0.012	**822.6**	0	673.5	0.012	**682**	0
180	15	786	0.011	**794.4**	0	773.2	0.013	**783.1**	0
185	15	888.7	0.016	**902.9**	0	835.7	0.015	**848.7**	0
190	15	927.3	0.002	**929.1**	0	648	0.011	**655.3**	0
195	15	823.6	0.024	**844**	0	854.8	0.020	**872.6**	0
200	15	866.8	0.020	**884.9**	0	781.1	0.020	**797.1**	0
80	20	**513.2**	0	512.4	0.002	439.5	0.005	**441.8**	0
85	20	389.5	0.004	**390.9**	0	**471.8**	0	470.6	0.003
90	20	493.1	0.005	**495.6**	0	424.3	0.002	**425.2**	0
95	20	564.2	0.008	**568.8**	0	**488.8**	0	488.3	0.001
100	20	**620.3**	0	618	0.004	**526.7**	0	525	0.003
105	20	593.5	0.003	**595.3**	0	576.3	0.002	**577.5**	0
110	20	673.7	0.009	**679.5**	0	612.8	0.009	**618.4**	0
115	20	662.2	0.020	**675.5**	0	622.1	0.005	**625.1**	0
120	20	549.5	0.015	**557.9**	0	594.2	0.005	**597.2**	0
125	20	**719.9**	0	719.1	0.001	**615.8**	0	614.3	0.002
130	20	689.6	0.010	**696.8**	0	**587.4**	0	581.5	0.010
135	20	**802.1**	0	792.8	0.012	607.2	0.006	**610.7**	0
140	20	**838.7**	0	838	0.001	762.8	0.014	**773.7**	0
145	20	825.3	0.019	**841.3**	0	548.2	0.018	**558.2**	0
150	20	758.9	0.007	**764.6**	0	763.3	0.016	**775.7**	0
155	20	**749.8**	0	749.2	0.001	712.8	0.021	**727.9**	0
160	20	**841.6**	0	834.9	0.008	868	0.007	**873.7**	0
165	20	912.1	0.019	**929.8**	0	742.6	0.025	**761.4**	0
170	20	783.6	0.010	**791.9**	0	669.3	0.021	**684**	0
175	20	911.1	0.004	**914.5**	0	756.9	0.020	**772**	0
180	20	868	0.007	**873.8**	0	873.2	0.015	**886.4**	0
185	20	994	0.004	**998.2**	0	930.1	0.013	**942.7**	0
190	20	1045.6	0.006	**1051.4**	0	742.8	0.023	**760.3**	0
195	20	954.3	0.005	**958.8**	0	956.4	0.024	**980.2**	0
200	20	994.5	0.005	**999.7**	0	900.5	0.002	**902.4**	0

Table 4. Empirical results on Combinatorial Auctions. Path distribution.

varying the number of bids from 80 to 200. For each parameter configuration, we generate samples of size 10. Table 4 shows the results for $z = \{15, 20\}$. Recall that for optimization tasks, only the SCP heuristic is defined.

The behavior for both configurations is almost the same. For 20 goods, the MBE fill-in outperforms the BE fill-in on 23 out of the 25 configurations of different number of bids when $z = 15$, and on 18 when $z = 19$. For 50 goods, the MBE fill-in outperforms the BE fill-in on 18 configurations of bids when $z = 15$, and on 20 when $z = 19$.

It is important to observe that the improvement over the BE fill-in is not as significant as for previous benchmarks. One possible reason is the nature of the marginalization operator: when summing, the quality of all operands impacts on the quality of the result; while when minimizing, the quality of the minimum operand is the only one that determines the quality of the result. Indeed, further investigation is needed.

5 Related Work

There are two early approaches based on mini-bucket elimination which use a variable elimination ordering different to the one used by bucket elimination: *greedy SIP* [10] and *Approximate Decomposition* (AD) [9].

Greedy SIP solves the problem by iteratively applying bucket elimination over subsets of functions. At each iteration, all variables are eliminated from the current subset and its elimination ordering is the one with induced width bounded by the control parameter z. The order in which variables are eliminated can be different from one iteration to another.

AD solves the problem by iteratively eliminating the variables of the problem and maintaining the width of the new problems bounded by z. If the elimination of a variable causes the width of the new problem to be greater than z, the new function is approximated with a combination of simpler ones such that the width is maintained under z.

Our scheme resembles these two approaches on that none of them uses the variable elimination ordering as dictated by bucket elimination. However, the value of our work is on clearly showing why a variable elimination heuristic should fit the actual structure of problems generated after each variable elimination.

6 Conclusions

Bucket Elimination (BE) and Mini-Bucket Elimination (MBE) are based on the sequential transformation of the original problem by sequentially eliminating variables, one at a time. The result of eliminating one variable is a new subproblem. Under the same variable elimination ordering, they generate a different sequence of subproblems. Although this important difference, MBE uses the elimination order obtained by a procedure designed for BE. Since this procedure selects the next variable to eliminate according to the structure of subproblems produced by BE, it will select *erroneous* variables according to the structure of subproblems generated by MBE.

This paper investigates a modification on how to compute the elimination ordering for MBE. Our approach computes the ordering by considering the real structure of the sequence of subproblems produced by MBE thanks to induced z-bounded hypergraphs. We demonstrate the effectiveness of this new ordering on a number of benchmarks over two important tasks: computing the probability of the evidence and finding the minimum cost assignment of a weighted CSP. We observed that the higher improvements are obtained on the first task. The nature of the marginalization operator may explain this fact. We plan to further investigate this issue.

There are other approximation algorithms based on variable elimination orderings (e.g., Iterative Join Graph Propagation [4]). In our future work we want to study the impact of our approach on their accuracy.

Acknowledgement

This work was supported by project TIN2009-13591-C02-01.

References

1. U. Bertelè and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.
2. R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.
3. R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, 2003.
4. R. Dechter, K. Kask, and R. Mateescu. Iterative join-graph propagation. In *Proceedings of the 18th Conference in UAI, Edmonton, Canada*, pages 128–136, 2002.
5. R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM*, 50(2):107–153, March 2003.
6. E. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29:24–32, March 1982.
7. K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129:91–131, 2001.
8. M.Pearson K.Leyton-Brown and Y.Shoham. Towards a universal test suite for combinatorial auction algorithms. *ACM E-Commerce*, pages 66–76, 2000.
9. D. Larkin. Approximate decomposition: A method for bounding and estimating probabilistic and deterministic queries. In *Proceedings UAI, Acapulco, Mexico*, pages 346–353, 2003.
10. D. Larkin. Semi-independent partitioning: A method for bounding the solution to cop's. In *9th International Conference on Principles and Practice of Constraint Programming*, pages 894–898, 2003.
11. J. D. Park. Using weighted max-sat engines to solve mpe. In *Proc. of the 18th AAAI*, pages 682–687, Edmonton, Alberta, Canada, 2002.
12. E. Rollon and R. Dechter. New mini-bucket partitioning heuristics for bounding the probability of evidence. In *Proc. of the* 24^{th} *AAAI, Atlanta, Georgia, USA*, 2010.
13. P. Shenoy. Axioms for dynamic programming. In *Computational Learning and Probabilistic Reasoning*, pages 259–275, 1996.
14. B. Wemmenhove, J. M. Mooij, W. Wiegerinck, M. A. R. Leisink, H. J. Kappen, and J. P. Neijt. Inference in the promedas medical expert system. In 11^{th} *Conference on Artificial Intelligence in Medicine, Amsterdam, The Netherlands*, pages 456–460, 2007.