Depth-First Mini-Bucket Elimination

Emma Rollon and Javier Larrosa

Universitat Politecnica de Catalunya,
Jordi Girona 1-3, 08034 Barcelona, Spain
erollon@lsi.upc.edu, larrosa@lsi.upc.edu

Abstract. Many important combinatorial optimization problems can be expressed as *constraint satisfaction problems* with *soft constraints*. When problems are too difficult to be solved exactly, approximation methods become the best option. *Mini-bucket elimination* (MBE) is a well known approximation method for combinatorial optimization problems. It has a control parameter z that allow us to trade time and space for accuracy. In practice it is the space and not the time that limits the execution with high values of z. In this paper we introduce a set of improvements on the way MBE handles memory. The resulting algorithm dfMBE may be orders of magnitude more efficient. As a consequence, higher values of z can be used which, in turn, yields significantly better bounds. We demonstrate our approach in scheduling, probabilistic reasoning and resource allocation problems.

1 Introduction

Constraint satisfaction problems (CSPs) involve the assignment of a set of variables subject to a set of constraints. The addition of *soft constraints* [1] extend the CSP framework to optimization tasks (we will assume optimization as minimization). Many problems in a variety of domains such as *probabilistic reasoning* [2], *bioinformatics* [3], *scheduling* [4], *etc*, can be naturally expressed as soft CSPs. In recent years a big effort has been made in the development of algorithms to solve this type of problems. In some cases, specialized algorithms can be designed to solve more efficiently particular problems. Nevertheless in this paper we will focus on general techniques.

There are two main approaches to solving soft CSPs: search and inference. *Search* algorithms traverse the tree of possible assignments typically following the deph-first branch-and-bound (BnB) principle [5, 6]. *Inference* algorithms solve the problem by a sequence of reductions following the dynamic programming principle [7]. A well known inference algorithm is *bucket elimination* (BE) [8]. It falls into the category of the so-called *decomposition methods* [7]. In general, solving a soft CSP is NP-hard. Therefore, all known algorithms require exponential resources in the worst case, which means that many instances cannot be solved with current technology. Given the practical importance of this type of problems, when exact methods fail, algorithms that approximate the solution become very desirable.

When an instance is too difficult to be solved exactly, both search and inference, can be adapted to approximate its solution. BnB maintains during the search an upper bound of the optimum. Thus, it can be seen as an any-time algorithm providing increasingly better upper bounds. Alternatively, if BnB is executed in an iterative deepening manner it provides an increasing sequence of lower bounds. Both of these approaches are polynomial in space, and the time can be adjusted to the user needs. The BE algorithm can also be approximated which results in the *mini-buckets elimination* algorithm (MBE) [9]. In its more general formulation, the outcome of MBE is a lower and an upper bound of the optimum. It is arguably one of the best-known general approximation algorithms for soft CSPs and it has shown to be effective in a variety of domains including several probabilistic tasks in bayesian networks. It uses a control parameter z that allow us to trade time and space for accuracy. The time and space complexity of MBE is exponential in z and it is important to note that, with current computers, it is the space, rather than the time, that prohibits the execution of the algorithm beyond certain values of z.

In this paper, we show how to decrease the space demands of MBE. Our approach is based on the concept of *computation tree* (CT). A CT provides a graphical view of the MBE execution and can be computed as a pre-process. It is somewhat similar to the tree-decomposition in decomposition methods [9], where the first step is to build the tree-decomposition and the second step is to solve the problem. Our first contribution is a set of local transformations to the CT with which a more rational use of memory is achieved. They include: *i*) *branch re-arrangement* (nodes are moved upwards along a branch which means that the elimination of a variable is anticipated) and, *ii*) *vertical tree compaction* (adjacent nodes are joined which means that a sequence of operations is performed in a single step).

The second contribution is the exploitation of *memory deallocation* of intermediate functions when they become redundant. By construction of CT, MBE can be seen as a top-down traversal of the CT. The order of the traversal is imposed by the order in which variables are eliminated. We make the observation that any top-down traversal of the CT would produce the same outcome. Then, we propose to traverse the CT in a *depth-first* manner in order to decrease the number of intermediate functions that must be simultaneously stored. We show that with a depth-first traversal of the CT, the order of children has an impact in the space complexity which provides an additional source of improvement. We also discuss the benefits of *horizontal node compaction.* It is important to note that none of these transformations risk the accuracy of the algorithm.

The new algorithm that incorporates all these techniques is called depth-first mini-bucket elimination dfMBE. Our experiments show in a number of domains that dfMBE may provide important space savings. The main consequence is that in a given computer (namely, for a fixed amount of memory), dfMBE(z) can be executed with a higher value of z than MBE(z) which, in turn, may yield better lower bounds.

The structure of the paper is as follows: Section 2 provides preliminary definitions, Section 3 introduces CTs and two local transformations, Section 4 introduces dfMBE and additional CT transformations, Section 5 reports our experimental results and, finally, Section 6 gives conclusions and discusses future work.

2 Preliminaries

2.1 Soft CSPs

Let $\mathcal{X} = (x_1, \ldots, x_n)$ be an ordered set of variables and $\mathcal{D} = (D_1, \ldots, D_n)$ an ordered set of domains. Domain D_i is a finite set of potential values for x_i. The assignment (i.e, instantiation) of variable x_i with $a \in D_i$ is noted $(x_i \leftarrow a)$. A *tuple* is an ordered set of assignments to different variables $(x_{i_1} \leftarrow a_{i_1}, \ldots, x_{i_k} \leftarrow a_{i_k})$. The set of variables $(x_{i_1}, \ldots, x_{i_k})$ assigned by a tuple t, noted $var(t)$, is called its *scope*. The size of $var(t)$ is the *arity* of t. We focus on two basic operations over tuples: The *projection* of t over $A \subseteq var(t)$, noted $t[A]$, is a sub-tuple of t containing only the instantiation of variables in A. 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. Projecting a tuple t over the empty set $t[\emptyset]$ produces the empty tuple λ. We say that a tuple t is a *complete instantiation* when $var(t) = \mathcal{X}$. Sometimes, when we want to emphasize that a tuple is a complete instantiation we will call it X.

A *soft CSP* is a triple $(\mathcal{X}, \mathcal{D}, \mathcal{F})$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $\mathcal{D} = \{D_1, \ldots, D_n\}$ are sets of variables and domains. $\mathcal{F} = \{f_1, \ldots, f_r\}$ is a set of functions that form an objective function. *Function* f_i over $S \subseteq \mathcal{X}$ associates valuations to tuples t such that $var(t) = S$. The set of variables S is the *scope* of f and is noted $var(f_i)$. Abusing notation, when $var(f_i) \subset var(t)$, $f_i(t)$ will mean $f_i(t[var(f_i)])$. Functions may be given explicitly as tables or implicitly as mathematical expressions or computer procedures. The space complexity of explicitly storing a function f_i is $sp(f_i) = \prod_{x_j \in var(f_i)} |D_j|$.

Different soft CSP frameworks differ in: the set of possible valuations, the way functions are combined in order to form the objective function, and the task required of the objective function [1]. For simplicity, in the following we will assume weighted CSPs. In the weighted CSP (WCSP) model, valuations are natural numbers, the objective function is the sum of the functions,

$$F(X) = \sum_{i=1}^{r} f_i(X)$$

and it has to be *minimized*.

2.2 Bucket and Mini-Bucket Elimination

Bucket elimination (BE)[8, 10] is a well-known algorithm for soft CSPs. It uses the following two operations on functions:

function $\texttt{BE}(\mathcal{X}, \mathcal{D}, \mathcal{F})$
1. **for each** $i = n..1$ **do**
2. $\quad\mathcal{B}_i := \{f \in F | \ x_i \in var(f)\}$
3. $\quad g_i := (\sum_{f \in \mathcal{B}_i} f) \downarrow x_i$;
4. $\quad F := (F \cup \{g_i\}) - \mathcal{B}_i$;
5. **endfor**
6. $t := \lambda$;
7. **for each** $i = 1..n$ **do**
8. $\quad v := \text{argmin}_{a \in D_i}\{(\sum_{f \in \mathcal{B}_i} f)(t \cdot (x_i \leftarrow a))\}$
9. $\quad t := t \cdot (x_i \leftarrow v)$;
10. **endfor**
11. **return**(g_1, t);

endfunction

Fig. 1. Bucket Elimination. Given a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a constant function g_1 (i.e, $var(g_1) = \emptyset$) with the optimal cost, along with one optimal assignment t.

- The *sum* of two functions f and g denoted $(f + g)$ is a new function with scope $var(f) \cup var(g)$ which returns for each tuple the sum of costs of f and g,
$$(f + g)(t) = f(t) + g(t)$$
- The *elimination* of variable x_i from f, denoted $f \downarrow x_i$, is a new function with scope $var(f) - \{x_i\}$ which returns for each tuple t the minimum cost extension of t to x_i,
$$(f \downarrow x_i)(t) = \min_{a \in D_i}\{f(t \cdot (x_i \leftarrow a))\}$$
where $t \cdot (x_i \leftarrow a)$ means the extension of t so as to include the assignment of a to x_i. Observe that when f is a unary function (*i.e.*, arity one), eliminating the only variable in its scope produces a constant.

The result of summing functions or eliminating variables cannot, in general, be expressed intensionally. Therefore, we store functions as tables.

BE (Figure 1) uses an arbitrary variable ordering o that we assume, without loss of generality, lexicographical (i.e, $o = (x_1, x_2, \ldots, x_n)$). It works in two phases. In the first phase (lines 1-5), the algorithm eliminates variables one by one, from last to first, according to o. The elimination of variable x_i is done as follows: $\mathcal{F}$ is the set of current functions. The algorithm computes the so called *bucket* of x_i, noted $\mathcal{B}_i$, which contains all cost functions in $\mathcal{F}$ having x_i in their scope (line 2). Next, BE computes a new function g_i by summing all functions in $\mathcal{B}_i$ and subsequently eliminating x_i (line 3). Then, $\mathcal{F}$ is updated by removing the functions in $\mathcal{B}_i$ and adding g_i (line 4). The new $\mathcal{F}$ does not contain x_i (all functions mentioning x_i were removed) but preserves the value of the optimal cost. The elimination of the last variable produces an empty-scope function (*i.e.*, a constant) which is the optimal cost of the problem. The second phase (lines 6-11) generates an optimal assignment of variables. It uses the set of buckets that were computed in the first phase. Starting from an empty assignment t (line 6), variables are assigned from first to last according to o. The optimal value for x_i is the best value regarding the extension of t with respect to the sum of functions in $\mathcal{B}_i$ (lines 8,9). We use argmin to denote the argument producing the minimum valuation. The time and space complexity of *BE* is exponential in a structural parameter called *induced width*. In practice, it is the space and not the time what makes the algorithm unfeasible in many instances.

Mini-bucket elimination (MBE) [9] is an approximation of BE that can be used to bound the optimum when the problem is too difficult to be solved exactly. Given a control parameter z, MBE partitions buckets into smaller subsets called mini-buckets such that their arity is bounded by $z+1$. Each mini-bucket is processed independently. At the end of the first phase, MBE has a lower bound of the problem optimum. In the second phase MBE computes a (non-necessarily optimal) assignment t. The evaluation of t on the objective function $F(t)$ constitutes an upper bound of the problem optimum. The pseudo-code of MBE is the result of replacing lines 3 and 4 in the algorithm of Figure 1 by,

3. $\{\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}\} := \text{Partition}(\mathcal{B}_i)$;
3b. **for each** $j = 1..k$ **do** $g_{i_j} := (\sum_{f \in \mathcal{P}_{i_j}} f) \downarrow x_i$;
4. $F := (F \cup \{g_{i_1}, \ldots, g_{i_k}\}) - \mathcal{B}_i$;

The time and space complexity of MBE is exponential in z. Parameter z allows to trade time and space for accuracy, because greater values of z increment the number of functions that can be included in each mini-bucket. Therefore, the bounds will be presumably tighter.

Consider as a example a WCSP instance with seven variables and the following set of cost functions,

$$\mathcal{F} = \{f_1(x_6, x_5, x_4), f_2(x_6, x_5, x_3), f_3(x_5, x_3, x_2), f_4(x_6, x_4, x_2), f_5(x_7, x_2, x_1), f_6(x_7, x_6, x_1)\}$$

One possible execution of MBE(3) along the lexicographical variable ordering leads to the following trace,

$Bucket_7$:	$f_6(x_7, x_6, x_1),\ f_5(x_7, x_2, x_1)$
$Bucket_6$:	$g_{7_1}(x_6, x_2, x_1) = (f_6 + f_5) \downarrow x_7$, $f_4(x_6, x_4, x_2),\ f_2(x_6, x_5, x_3),\ f_1(x_6, x_5, x_4)$
$Bucket_5$:	$g_{6_1}(x_5, x_4, x_3) = (f_1 + f_2) \downarrow x_6,\ f_3(x_5, x_3, x_2)$
$Bucket_4$:	$g_{6_2}(x_4, x_2, x_1) = (f_4 + g_{7_1}) \downarrow x_6$, $g_{5_1}(x_4, x_3, x_2) = (f_3 + g_{6_1}) \downarrow x_5$
$Bucket_3$:	$g_{4_1}(x_3, x_2, x_1) = (g_{6_2} + g_{5_1}) \downarrow x_4$
$Bucket_2$:	$g_{3_1}(x_2, x_1) = g_{4_1} \downarrow x_3$
$Bucket_1$:	$g_{2_1}(x_1) = g_{3_1} \downarrow x_2$
Result:	$g_{1_1}() = g_{2_1} \downarrow x_1$

3 Improving MBE memory usage

The first phase of MBE (as well as BE) can be seen as an algebraic expression that combines sums and variable eliminations. For instance, the execution of MBE(3) in the previous example is equivalent to the computation of the following expression,

$$((f_3 + (f_1 + f_2) \downarrow x_6) \downarrow x_5 + (f_4 + (f_6 + f_5) \downarrow x_7) \downarrow x_6) \downarrow x_4 \downarrow x_3 \downarrow x_2 \downarrow x_1$$

Note that each function appears only once in the formulae.

A *computation tree* (CT first introduced in [11]) provides a graphical view of the algebraic expression. The leaves are the original functions (arguments of the formulae) and internal nodes represent the computation of intermediate functions. If the node has only one child, the only operation performed is the elimination of one variables. Otherwise, all the children are summed and one variable is eliminated. Figure 2.A depicts the CT of the previous example. Dotted lines emphasize tree-leaves, which are associated to original functions. Even when original function are given explicitely as tables, do not include their space in the MBE cost. Adjacent to each internal node we indicate the variable that is eliminated. Although CTs are somehow related to decomposition-trees, they differ in the way they represent original functions. Besides, since CTs originate from MBE executions, they do not need to satisfy the running intersection property [7].

In the following we distinguish the computation of the CT from the evaluation of its associated expression. Given the scope of the original functions, a variable ordering, a policy for mini-bucket partitioning and a value for z, it is possible to compute the corresponding CT as a pre-process. Computing the CT is no more than finding the set of computations that the algorithm will perform in order to evaluate the formula.

One advantage of computing the CT in a pre-process is that it makes it easy to obtain the exact memory demands of MBE by summing the space requirements of every internal node of the CT. For instance, the CT in Figure 2.A, will need to store 5 functions of arity 3, 1 functions of arity 2, 1 function of arity 1 and 1 function of arity 0. Assuming domains of size 10, MBE will need to store $5 \times 10^3 + 1 \times 10^2 + 1 \times 10^1 + 1 \times 10^0 = 5111$ table entries.

CTs allow us to identify and remedy some space inefficiencies of MBE. In the following we describe two local transformations of CTs that improve their space requirements.

3.1 Branch Re-arrangement

Consider again the CT in Figure 2.A. Observe that if we follow any branch in top-down order, variables are eliminated in decreasing order, because this is the order used by MBE. As a consequence, the elimination of x_1 is left to the end. However, this variable only appears in the two leftmost leaves. It is inefficient to carry it over down to the CT root, since it could have been eliminated higher up.

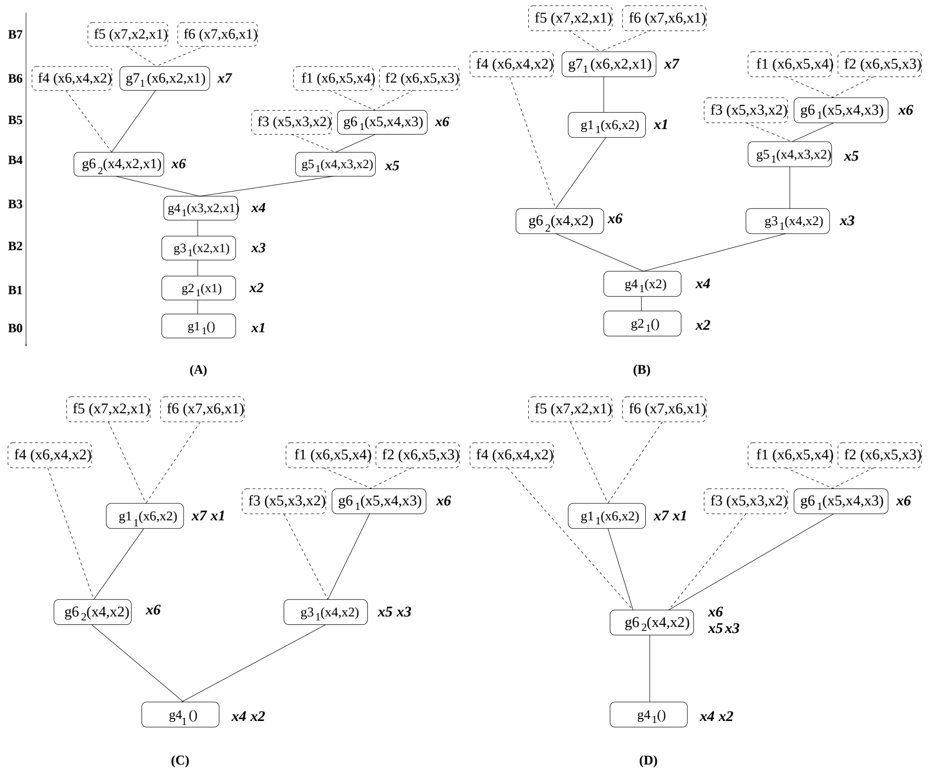

Fig. 2. Four different computation trees: A) original CT, B) after branch re-arrangement, C) after vertical compaction, D) after horizontal compaction

Consider a node v of a CT with a single child. Let x_i be the variable that is eliminated at v. Let u be the first descendent of v with $k > 1$ children. If only one child w of u has x_i in its scope, node v (namely, the elimination of x_i) can be moved in between w and u. We only perform the change if w is not a leaf. *Branch re-arrangement* is the process of applying the previous rule in a bottom-up order, moving nodes as close to the leaves as possible. The benefit of branch re-arrangement is that x_i disappears from the scope of intermediate functions earlier in the tree. In the CT of Figure 2.A, the leftmost branch can be re-arranged: variable x_1 can be eliminated right after x_6. Moreover, the rightmost branch can also be re-arranged: variable x_3 can be eliminated right after x_4. Figure 2.B shows the resulting CT. The space requirements of the new CT are decreased from 5111 to 3311. Observe that branch re-arrangement can never increase the space requirements of a CT.

3.2 Vertical Compaction

Consider the CT in Figure 2.B. There are two single-child nodes. In single-child nodes the only associated computation is a variable elimination. MBE considers each variable elimination as an independent operation because they take place in different buckets. However, a sequence of variable eliminations can be performed simultaneously in a single step without changing the outcome or increasing the time complexity. The advantage is that intermediate functions do not need to be stored.

Vertical compaction is the process of merging *internal linear paths* into one node representing the sequence of computations. An internal linear path is a path between an internal node v and one of its ancestors w, $(v, v_1 \ldots, v_k, w)$, such that every node in the path except v has only one child. After the compaction every internal node of the CT has $k > 1$ children. There is one exception: there may be internal nodes with only child if the child is a leaf. Figure 2.C depicts the result of applying vertical compaction to the CT of Figure 2.B. The space requirements of the new CT are decreased from 3311 to 1301. It is clear that the compaction of a CT may produce space saving and can never increase the space requirements of a CT.

4 Depth-First MBE

A CT can be traversed in any top-down order. A node can be computed as soon as all its children are available. Whatever traversal strategy is used it has to keep all intermediate functions because they are used in the second phase of the algorithm in order to compute the upper bound. However, the space consumption of the traversal can be drastically reduced if we sacrifice the upper bound and deallocate the memory used by intermediate functions when they become redundant. A function becomes redundant as soon as its parent has been computed. Note that an alternative solution that we do not explore in this paper is to store redundant functions in the hard-disk. Thus, the upper bound is not lost.

Without memory deallocation the traversal order has no effect on the space complexity, but this is no longer true when memory is deallocated. Traversing the CT depth-first has the advantage of only demanding the space of the current branch: computing a node only requires to have available its children, so they have to be sequentially and recursively computed. We denote by dfMBE the algorithm that traverses depth-first the CT and deallocates memory when intermediate functions become redundant. The space complexity of dfMBE can be formalized by means of a recurrence. Let v be a node, g_v the associated function and $(w_1, \ldots, w_k)$ its ordered set of children. $R(v)$ is the space complexity of computing the sub-tree rooted by a CT node v and is given by,

$$R(v) = \max_{i=1}^{k+1}\{\sum_{j=1}^{i-1} sp(g_{w_j}) + R(w_i)\}$$

where $R(w_{k+1}) = sp(g_v)$ by definition. Also, the space $sp()$ of original functions is 0 because we do not count it as used by the algorithm. The space complexity of dfMBE is obtained by evaluating $R(v)$ at the root of the CT. In words, the recursion indicates that the space required to compute node v is the maximum among the space required to compute its children. However, when computing a given child, the space occupied by all its previous siblings must be added because they need to be available for the final computation of v.

Consider the CT of Figure 2.C. We showed in the previous Section that, with no memory deallocation, the space cost of internal nodes was 1301. If the CT is traversed depth-first, the cost (disregarding original functions) is,

$$\max\{R(g_{6_2}), sp(g_{6_2}) + R(g_{3_1}), sp(g_{6_2}) + sp(g_{3_1}) + sp(g_{4_1})\} =$$

$$\max\{200, 100 + 1100, 100 + 100 + 1\} = 1200$$

Observe that the order of children affects the space complexity of dfMBE. For instance, if we reverse the two children of the root in Figure 2.C, the space complexity of dfMBE is decreased to,

$$\max\{R(g_{3_1}), sp(g_{3_1}) + R(g_{6_2}), sp(g_{3_1}) + sp(g_{6_2}) + sp(g_{4_1})\} =$$

$$\max\{1100, 100 + 200, 100 + 100 + 1\} = 1100$$

In our implementation of dfMBE we make an additional optimization of the CT by processing nodes from leaves to the root. At each node, we swap the order of two of its children if it brings a space improvement.

Consider now the two children of the root-node in the CT of Figure 2.C. The scope of the associated functions g_{6_1} and g_{6_2} is the same. Since they will be summed up, one table can be shared to stored both of them as follows: the table entries are initialized to 0, the two functions are computed sequentially, and each function value is added to the table current value. Figure 2.D illustrates this idea. The cost of dfMBE with this new CT is,

$$\max\{R(g_{6_2}), sp(g_{6_2}) + sp(g_{4_1})\} =$$

$$\max\{\max\{100, 100 + 100, 100 + 1000, 100 + 1000\}, 100 + 1\} = 1100$$

which brings no gain over the CT in Figure 2.C. However, in some cases it may bring significant benefits. Note that $R(g_{6_2}) = \max\{R(g_{1_1}), sp(g_{1_1}) + sp(g_{6_2}), sp(g_{6_2}) + R(g_{6_1}), sp(g_{6_2}) + sp(g_{6_1})\}$. In our implementation, we check siblings pair-wise. If sharing their storing table produces space savings we take such an action.

5 Experimental Results

We have tested our approach in three different domains. We compare the memory requirements for MBE, MBE' (i.e, mini-buckets under the computation tree resulting from branch re-arrangement and vertical compaction), and dfMBE in a given computer (in other words, with a fixed amount of memory). For each domain we execute MBE(z_1), MBE'(z_2) and dfMBE(z_3), where z_1, z_2 and z_3 are the highest feasible values of the control parameter for each algorithm, given the available memory.

In all our experiments, the original CT was obtained assuming a MBE execution in which the order of variable elimination was established with the *min-degree* heuristic. For the elimination of each variable, mini-buckets are constructed one by one with the following process: Select one original function (or a non-original function if there are no original functions left). Choose among the remaining functions the one that adds the least number of variables to the mini-bucket until no more functions can be included in that mini-bucket.

In our benchmarks domain sizes range from 2 to 44, and some instances have variables with different domain size. Consequently, the arity of a function is not a proper way to indicate its spacial cost, which means that the control parameter z of MBE may be misleading (it forbids a function of arity $z + 1$ with binary domains and allows a function of arity z with domains of size 4 that is much more costly to store). We overcome this problem by modifying the meaning of z: In the original formulation of MBE, the *arity* of intermediate functions is bounded by z, but in our implementation the *size* of intermediate functions is bounded by 2^z.

5.1 Scheduling

For our first experiment, we consider the scheduling of an earth observation satellite. Given a set of candidate photographs, the problem is to select the best subset that the satellite will actually take. The selected subset of photographs must satisfy a large number of imperative constraints and at the same time maximize the importance of selected photographs. We experimented with instances from the Spot5 satellite [4] that can be trivially translated into the WCSP framework. These instances have unary, binary and ternary constraints, and domains of size 2 and 4. Some instances include in their original formulation an additional capacity constraint imposed by the on-board storage limit. In our experiments we discarded such constraint.

Figure 3 reports the results that we have obtained assuming a computer with a memory limit of 1.5 Gigabytes. The first column identifies the instance. The second column indicates the induced width with the min-degree ordering. The third, fourth and fifth columns report the memory requirements in Megabytes with the three algorithms for different values of z. If the number is given in italics it means that it surpasses the space limit of the computer and the algorithm could not be executed (the memory requirement was obtained from the analysis of the CT). The sixth and seventh column indicate the value of z and the lower bound that is obtained. For each instance, we report results for three increasing values of z: the limit for MBE, MBE' and dfMBE. It can be observed that MBE' requires from 2 to 10 times less memory than MBE, which allows the execution with values of z up to 4 units higher. However, the most impressive results are obtained with dfMBE, which may require 275 times less space than MBE (e.g. instance 1405). As a consequence dfMBE can be executed with values of z up

Instance	w^*	Memory Requirement (Mb)			z	Lower Bound
		CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}		
1504	43	*17161*	*10373*	1052	27	158274
		1945	911	65	23	148259
		1240	577	34	22	142257
1506	51	*227707*	*50435*	1310	27	180356
		5503	1099	24	21	180316
		1185	214	4	18	166305
1401	156	*137825*	*11250*	524	26	210085
		11430	874	40	22	203083
		1286	131	6	19	196080
1403	156	*237480*	*28144*	1048	27	223189
		13416	1277	36	22	193185
		1153	125	5	18	189180
1405	156	*325213*	*54378*	1179	27	219302
		7226	1317	24	21	214283
		1548	289	3	18	203268
28	139	*113739*	*14764*	1048	27	141105
		8374	1424	65	23	141105
		694	109	5	19	148105
42	51	*22558*	*6032*	1572	28	125050
		2112	1123	147	24	135050
		1090	590	65	23	133050
5	83	*82823*	*38425*	917	27	206
		1861	843	16	21	192
		536	253	4	18	186
408	60	*17903*	*7966*	1048	27	5197
		2609	1355	163	24	6195
		1408	752	5	23	5197
412	61	*58396*	*24513*	1179	27	14258
		2771	882	40	22	17224
		1420	434	16	21	14220
414	144	*172071*	*24566*	1048	27	19295
		8605	1205	49	22	18301
		1154	166	4	19	18292
505	39	*15833*	*8644*	1067	27	18231
		2834	1534	139	24	19217
		1488	800	65	23	19206
507	91	*76346*	*16932*	1310	27	15286
		6222	1571	81	23	15280
		1217	250	10	20	12255
509	151	*130553*	*26671*	1114	27	18286
		6812	1008	40	22	17285
		946	162	4	19	17267

Fig. 3. Spot5 results. Memory bound of 1.5 Gb.

to 9 units higher (e.g. instance 1506), which in turn yields lower bounds up to 20% higher (e.g. instance 507). The mean space gain from MBE to dfMBE is 113.34, the mean increment of z is 7 and the mean increment of the lower bound is 8.74%.

5.2 Probabilistic Reasoning

Bayesian Networks provides a formalism for reasoning about partial beliefs under conditions of uncertainty [2]. They are defined by a directed acyclic graph over nodes representing variables of interest. The arcs indicate the existence of direct causal influences between linked variables quantified by conditional probability tables (CPTs) that are attached to each cluster of parents-child nodes in the network. There are several possible tasks over a belief network. We tested the performance of our scheme for solving the *most probable explanation* (MPE) task: given evidence $x_1 \leftarrow v_1, \ldots, x_p \leftarrow v_p$ (i.e., some variable assignments), its MPE is the *maximization* of the objective function,

$$P(X) = \prod_{i=1}^{r} f_i(X)$$

subject to tuples that respect the evidence. It is easy to see that MPE can be expressed as a WCSP by replacing probability tables by their logarithm. We use two types of belief networks: Random and Noisy-OR Networks [12].

Uniform random bayesian networks and *noisy-OR* networks are generated using parameters (N, K, C, P), where N is the number of variables, K is their domain size, C is the number of conditional probability tables, and P is the number of parents in each conditional probability table. Instances are generated by selecting C variables at random. For each selected variable x_i, P parents are randomly selected from the set of variables with index less than i (if $i \leq P$ only $i-1$ parents are selected).

For random bayesian networks, each probability table is randomly generated using a uniform distribution. For noisy-OR networks, each probability table represents a noisy OR-function. For each CPT, we randomly assign to each parent variable y_j a value $P_j \in [0..P_{noise}]$. The CPT is then defined as, $P(x = 0|y_1, \ldots, y_P) = \prod_{y_j=1} P_j$ and $P(x = 1|y_1, \ldots, y_P) = 1 - P(x = 0|y_1, \ldots, y_P)$.

Table 4 present results of random and noisy-OR networks assuming a computer with a memory limit of 512 Megabytes. In each table we fix parameters N, K and P and change the value of C in order to control the network's sparseness. For each parameter setting we generate and solve a sample of 20 instances. We always assumed empty evidence and report mean values.

It can be observed that dfMBE requires from 15 to 29 times less memory than MBE, which allows the execution with values of z up to 3 units higher. The mean space gain from MBE to dfMBE is 18.56, the mean increment of z is 3.51 and the mean increment of the lower bound is 5.75%. For uniform random networks we also report the mean number of instances executed with $CT_{MBE'}$ and CT_{dfMBE} in which the lower bound increases with respect its execution with CT_{MBE} and $CT_{MBE'}$, respectively (i.e., %*better* column).

With random networks we also executed the efficient WCSP branch-and-bound solver TOOLBAR [1] initializing its upper bound with the lower bound given by dfMBE and observed that it did not terminate with a time limit of one hour. Considering that dfMBE with the highest z value takes less than 300 seconds in this domain, we conclude that dfMBE is a better approach than iterative deepening branch and bound.

We observed that noisy-OR networks could be easily solved to optimality with TOOLBAR. Therefore, we also report for each parameter setting and each value of z, how many instances are solved to optimality with MBE, MBE' and dfMBE.

[1] http://mulcyber.toulouse.inra.fr/projects/toolbar/

Uniform Random Bayesian Networks							
		Memory Requirement (Mb)				Lower	
N, C, P	w^*	CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}	z	Bound	% better
128, 85, 4	31.71	*3635*	*598*	239	26.36	18.61	40
		2579	315	171	25.75	18.35	90
		370	84	45	22.95	17.56	-
128, 95, 4	43.96	*4144*	*999*	205	26.21	20.68	50
		1941	317	146	24.9	20.34	90
		335	94	43	22.5	19.51	-
128, 105, 4	38.71	*4537*	*825*	264	26.2	23.58	60
		2192	391	185	25.3	23.27	95
		358	89	48	22.7	22.16	-
128, 115, 4	48.32	*4114*	807	261	25.85	26.22	60
		1823	345	172	24.7	25.61	100
		355	99	43	22.5	24.69	-

Noisy-OR $P_{noise} = 0.40$						
		Memory Requirement (Mb)				
N, C, P	w^*	CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}	z	% solved
128, 85, 4	35.39	*4777*	*662*	164	26.35	73
		2805	256	153	25.6	68
		331	68	29	22.65	47
128, 95, 4	38.61	*4331*	*681*	222	26.25	84
		2545	308	169	25.35	84
		340	74	34	22.55	58
128, 105, 4	43.06	*3125*	*683*	260	25.55	50
		1646	285	136	24.6	50
		364	91	45	22.45	15
128, 115, 4	46.51	*4446*	*918*	199	25.95	65
		1530	352	149	24.75	50
		340	102	46	22.55	25
Noisy-OR $P_{noise} = 0.50$						
128, 85, 4	40.74	*4780*	*631*	242	26.45	75
		3154	330	177	25.7	75
		384	71	33	22.8	60
128, 95, 4	38.12	*3663*	356	243	25.89	55
		2170	309	158	25.15	55
		368	76	49	22.63	25
128, 105, 4	43.04	*5080*	*952*	245	26.4	65
		2006	329	109	24.8	65
		371	79	33	22.6	45
128, 115, 4	46.25	*3506*	*964*	227	26.05	60
		1552	342	176	24.7	45
		384	94	43	22.5	35

Fig. 4. MPE on bayesian networks. 20 samples. Memory bound of 512 Mb.

5.3 Resource allocation

For our third experiment, we consider the *frequency assignment problem* where the task is to assign non-interfering frequencies to a set of communication links. We experimented with some instances of the so-called *radio link frequency assignment problem* [13] that can be expressed as WCSP. The optimization task is to provide the assignment with minimum global interference. In its usual formulation, these instances have binary cost functions and domains of size up to 44. We experimented with CELAR6, CELAR7 and graph instances. For lack of space, Table 5 only reports graph instances where we obtained the best results. It can be observed that dfMBE is also very effective in this domain. It can require on average by 430.25 times less memory than MBE, which allows the execution with values of z up to 5.25 units larger.

Instance	w^*	Memory Requirement (Mb)			z
		CT_{MBE}	$CT_{MBE'}$	CT_{dfMBE}	
graph05	135	*15955*	*1992*	49	28
		12880	1102	86	27
		1483	201	25	24
graph06	296	*30364*	*2544*	300	28
		17291	1320	300	27
		1354	117	10	23
graph07	146	*14797*	*1866*	527	28
		8187	266	49	27
		1511	180	45	24
graph11reduc	275	*30331*	*2044*	113	28
		15630	1183	113	27
		1267	154	22	23
graph11	495	*55260*	*3079*	22	28
		5935	338	30	25
		547	83	11	21
graph12	234	*23532*	*3570*	692	28
		3399	493	134	26
		1379	230	21	24
graph13reduc	619	*67123*	*6447*	723	28
		9964	1070	121	25
		1572	141	13	22
graph13	706	*89091*	*6828*	1067	28
		7354	515	24	25
		806	161	11	21

Fig. 5. RLFAP. Memory bound of 1.5 Gb.

6 Conclusions and Future Work

Mini-bucket elimination (MBE) is a well-known approximation algorithm for combinatorial optimization problems. Its output is an upper and lower bound of the problem optimum. It has a control parameter with which the user can trade computing resources (namely, cpu time and memory) for approximation accuracy. With current computers it is usually the space rather than the cpu time what imposes a technological limit to the control parameter.

In this paper we have introduced a set of improvements to the spatial cost of MBE. Our approach is based on the concept of computation trees (CT), which provide a pictorical view of the MBE execution. We show that CTs uncover some space inefficiencies of MBE. Such inefficiencies can be overcome by local transformations of the CT that preserve the outcome of the algorithm and its time complexity. In particular, we introduce the concepts of branch re-arrangement and vertical compaction of CTs.

Besides, we show that if we sacrifice the upper bound we can deallocate intermediate computations that are very space consuming. In this context, we introduce depth-first MBE (dfMBE), that traverses the CT in a depth-first manner. The space demands of dfMBE can also be reduced by additional CT transformations such as children re-ordering and children merging.

We demonstrate the relevance of dfMBE in scheduling, probabilistic reasoning and resource allocation where we show that dfMBE can divide the space demand of MBE by a factor of 18.56 to 430.25 depending on the domain. Such space decrement allows the execution of dfMBE with higher values of the control parameter, which in turn, may yield better bounds.

In our future work we want to investigate more compact encodings for cost tables. A simple approach is to establish a default cost and store (e.g. in a hash table) only those tuples with non-default cost. A more sophisticated approach is to explore encodings based on Binary Decision Diagrams (BDDs), a graph based representation for boolean function manipulation. In this paper we have assumed a given mini-bucket partition policy. In our experiments, we observed that the chosen policy may have a big influence in both the topology of the CT and the quality of the reported bounds. We want to improve our understanding of such phenomenon and establish robust and effective policies.

References

1. Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-based CSPs and valued CSPs: Frameworks, properties and comparison. Constraints **4** (1999) 199–240
2. Pearl, J.: Probabilistic Inference in Intelligent Systems. Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA (1988)
3. Gilbert, D., Backofen, R., Yap, R., eds.: Constraints: an International Journal (Special Issue on Bioinformatics). Volume 6(2-3). Kluwer (2001)
4. Bensana, E., Lemaitre, M., Verfaillie, G.: Earth observation satellite management. Constraints **4(3)** (1999) 293–299
5. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: A survey. Operations Research **14(4)** (1966) 699–719
6. Larrosa, J., Schiex, T.: Solving weighted csp by maintaining arc-consistency. Artificial Intelligence **159** (2004) 1–26
7. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural csp decomposition methods. Artificial Intelligence **124** (2000) 243–282
8. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113** (1999) 41–85
9. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM **50** (2003) 107–153
10. Bertele, U., Brioschi, F.: Nonserial Dynamic Programming. Academic Press (1972)
11. Larrosa, J.: On the time complexity of bucket elimination algorithms. Technical report, University of California at Irvine (2001)
12. Kask, K., Dechter, R.: A general scheme for automatic generation of search heuristics from specification dependencies. Artificial Intelligence **129** (2001) 91–131
13. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.: Radio link frequency assignment. Constraints **4** (1999) 79–89