Mini-bucket Elimination with Bucket Propagation

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 new propagation phase that MBE should execute at each bucket. The purpose of this propagation is to jointly process as much information as possible. As a consequence, the undesirable lose of accuracy caused by MBE when splitting functions into different mini-buckets is minimized. We demonstrate our approach in *scheduling*, *combinatorial auction* and *max-clique* problems, where the resulting algorithm MBE^p gives important percentage increments of the lower bound (typically 50% and up to 1566%) with only doubling the cpu time.

1 Introduction

It is well recognized that many important problems belong to the class of combinatorial optimization problems. In general, combinatorial optimization problems are NP-hard. Therefore, they cannot be solved efficiently with current technologies. Then, the only thing that we can possibly do is to find near-optimal solutions. In that context, it is also desirable to have a quality measure of the solution. One way to achieve this goal is to provide a lower and an upper bound of the optimum. The smaller the gap, the closer we are to the true optimum.

Typically, best results are obtained developing *ad-hoc* techniques for the instances of interest. However, this requires a lot of work, including the time to learn specific domain peculiarities. An alternative, is to used generic techniques. Although they may not give so accurate results, it may be enough in some applications. Besides, they may provide the starting reference point to evaluate new *ad-hoc* techniques.

Mini-bucket Elimination (MBE) [1] is one of the most popular bounding techniques. Assuming minimization problems, MBE provides a lower bound of the optimum and can be combined with local search which provide upper bounds [1]. MBE is very general, since it can be applied to any problem that falls into the category of *graphical models*. Graphical models include very important optimization frameworks such as soft constraint satisfaction problems [2], Max-SAT, bayesian networks [3], etc. These frameworks have important applications in fields such as *routing* [4], *bioinformatics* [5], *scheduling* [6] or *probabilistic reasoning* [7]. The good performance of MBE in different contexts has been widely proved [1, 8, 7].

Interestingly, MBE has a parameter z which allow us to trade time and space for accuracy. With current computers, it is the space and not the time what bounds the maximum value of z that can be used in practice. In our previous work [9], we introduced a set of improvements on the way MBE handles memory. As a result, MBE became orders of magnitude more efficient. Thus, higher values of z can be used which, in turn, yields significantly better bounds. In this paper we continue improving the practical applicability of MBE. In particular, we introduce a new propagation phase that MBE must execute at each bucket. Mini-buckets are structured into a tree and costs are moved along branches from the leaves to the root. As a result, the root mini-bucket accumulates costs that will be processed together, while classical MBE would have processed them independently. Note that the new propagation phase does not increase the complexity with respect classical MBE.

Our experiments on *scheduling*, *combinatorial auctions* and *maxclique* show that the addition of this propagation phase increases the quality of the lower bound provided by MBE quite significatively. Although the increase depends on the benchmark, the typical percentage is 50%. However, for some instances, the propagation phase gives a dramatic percentage increment up to 1566%.

2 Preliminaries

2.1 Soft CSP

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, where D_i is the 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* t 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 *scope* of t, noted $var(t)$, is the set of variables that it assigns. The *arity* of t is $|var(t)|$. The *projection* of t over $Y \subseteq var(t)$, noted $t[Y]$, is a sub-tuple of t containing only the instantiation of variables in Y. 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 $var(t) = \mathcal{X}$. In the following, abusing notation, when we write $\forall_{t \in Y}$ we will mean $\forall_{t \texttt{ s.t. } var(t)=Y}$.

[1] In the original description MBE also provides an upper bound, but in this paper we will disregard this feature

Let A be an ordered set of values, called valuations, and $+$ a conmutative and associative binary operation $+: A \times A \rightarrow A$ such that exists an identity element 0 (namely, $\forall a \in A,\ a + 0 = a$), and satisfies monotonicity (namely, $\forall a, b, b \in A$, if $a \geq b$ then $(a + b \geq b + c)$).

$\mathcal{F} = \{f_1, \ldots, f_r\}$ is a set of functions. Each function f_j is defined over a subset of variables $var(f_j) \subseteq \mathcal{X}$ and returns values of A (namely, if $var(t) = var(f_j)$ then $f_j(t) \in A$). For convenience, we allow to evaluate $f_j(t)$ when $var(t) \supset var(f_j)$, being equivalent to $f_j(t[var(f_j)])$. In this paper we assume functions explicitly stored as tables.

A *soft CSP* is a triplet $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ where each function $f \in \mathcal{F}$ specifies how good is each different partial assignment of $var(f)$. The sum $+$ is used to *aggregate* values from different functions. The global quality of an assignment is the sum of values given by all the functions. The usual task of interest is to find the best complete assignment $\mathcal{X}$ in terms of A. Different soft CSP frameworks differ in the semantics of A. Well-known frameworks include *probabilistic* CSPs, *weighted* CSPs, *fuzzy* CSPs, etc [2].

A soft CSP framework is *fair* [10] if for any pair of valuations $\alpha, \beta \in A$, with $\alpha \leq \beta$, there exists a maximum difference of β and α. This unique maximum difference of β and α is denoted by $\beta - \alpha$. This property ensures the equivalence of the problem when the two operations $+$ and $-$ are applied. In [10] it is shown that the most important soft constraint frameworks are fair. Although our approach can be used in any fair soft constraint framework, for the sake of simplicity, we will focus on weighted CSPs. In weighted CSPs (WCSPs) A is the set of natural numbers, $+$ and $-$ are the usual sum and subtraction. Thus, the set of soft constraints define the following objective function to be minimized,

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

2.2 Operations over Functions

- 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 $t \in var(f) \cup var(g)$ the sum of costs of f and g,

$$(f + g)(t) = f(t) + g(t)$$

- Let f and g be two functions such that $var(g) \subseteq var(f)$ and $\forall t \in var(f),\ f(t) \geq g(t)$. Their *subtraction*, noted $f - g$ is a new function with scope $var(f)$ defined as,

$$(f - g)(t) = f(t) - g(t)$$

  for all tuple $t \in var(f)$.
- 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,

function BE($\mathcal{X},\mathcal{D},\mathcal{F}$)
1. **for each** $i = n..1$ **do**
2. $\quad\mathcal{B} := \{f \in F |\ x_i \in var(f)\}$
3. $\quad g := (\sum_{f \in \mathcal{B}} f) \downarrow x_i;$
4. $\quad F := (F \cup \{g\}) - \mathcal{B};$
5. **endfor**
6. **return**($\mathcal{F}$);

endfunction

Fig. 1. Bucket Elimination. Given a WCSP $(\mathcal{X},\mathcal{D},\mathcal{F})$, the algorithm returns $\mathcal{F}$ containing a constant function with the optimal cost.

$$(f \downarrow x_i)(t) = \min_{a \in D_i} \{f(t \cdot (x_i \leftarrow a))\}$$

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 *projection* of function f over $Y \subset var(f)$, denoted $f[Y]$, is a new function with scope Y which returns for each tuple t the minimum cost extension of t to $var(f)$,

$$(f[Y])(t) = \min_{t' \in var(f) \text{ s.t. } t' = t \cdot t''} f(t')$$

Observe that variable elimination and projection are related with the following property,

$$(f \downarrow x_i) = f[var(f) - \{x_i\}]$$

2.3 Bucket and Mini-Bucket Elimination

Bucket elimination (BE, Figure 1)[11, 12] is a well-known algorithm for weighted CSPs. It uses an arbitrary variable ordering o that we assume, without loss of generality, lexicographical (i.e, $o = (x_1, x_2, \ldots, x_n)$). 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 $\mathcal{B}$, which contains all cost functions in $\mathcal{F}$ having x_i in their scope (line 2). Next, BE computes a new function g by summing all functions in $\mathcal{B}$ and subsequently eliminating x_i (line 3). Then, $\mathcal{F}$ is updated by removing the functions in $\mathcal{B}$ and adding g (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 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) [1] 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 join arity is bounded by $z + 1$. Each mini-bucket is processed independently. Consequently, the output of MBE is a lower bound of the true optimum. The pseudo-code of MBE is the result of replacing lines 3 and 4 in the algorithm of Figure 1 by,

3. $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\} :=$ Partition($\mathcal{B}$);
3b. **for each** $j = 1..k$ **do** $g_j := (\sum_{f \in \mathcal{P}_j} f) \downarrow x_i$;
4. $F := (F \cup \{g_1, \ldots, g_k\}) - \mathcal{B}$;

The time and space complexity of MBE is $O(d^{z+1})$ and $O(d^z)$, respectively. Parameter z allow us 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. MBE constitutes a powerful yet extremely general mechanism for lower bound computation.

3 Equivalence-preserving transformations in fair frameworks

We say that two WCSPs are equivalent if they have the same optimum. There are several transformations that preserve the equivalence. For instance, if we take any pair of cost functions $f, g \in \mathcal{F}$ from a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ and replace them by their sum $f + g$, the result is an equivalent problem. The replacement of $\mathcal{B}$ by g performed by BE (Figure 1) is another example of equivalence-preserving transformation. Very recently, a new kind of WCSP transformation has been used in the context of soft local consistency [13, 14]. The general idea is to *move* costs from one cost function to another. More precisely, costs are subtracted from one cost function and added to another. Formally, let f and h be two arbitrary functions. The *movement of costs* from f to g is done sequentially in three steps:

$h := f[var(f) \cap var(g)]$
$f := f - h$
$g := g + h$

In words, function h contains costs in f that can be captured in terms of the common variables with g. Hence, they can be kept either in h or in f. Then, this costs are moved from f to g. The time complexity of this operation is $O(d^{\max\{|var(f)|, |var(g)|\}})$. The space complexity is the size of h stored as a table, $O(d^{|var(h)|})$, which is negligible in comparison with the larger function f.

Example 1. Consider the functions on Figure 2 (a). They are defined over boolean domains and given as a table of costs. Let function h represents the costs that g:

x_i	x_j	
f	f	5
f	t	4
t	f	1
t	t	6

f:

x_i	x_k	
f	f	2
f	t	3
t	f	4
t	t	5

(a)

g:

x_i	x_j	
f	f	7
f	t	6
t	f	5
t	t	10

f:

x_i	x_k	
f	f	0
f	t	1
t	f	0
t	t	1

(b)

$g \downarrow x_i$

x_j	
f	1
t	4

$f \downarrow x_i$

x_k	
f	2
t	3

(c)

$g \downarrow x_i$

x_j	
f	5
t	6

$f \downarrow x_i$

x_k	
f	0
t	1

(d)

Fig. 2. Example of functions.

can be moved from function f to function g. Observe that, as f and g only share variable x_i, then $h = f[x_i]$, where $h(false) = 2$ and $h(true) = 4$. Figure 2 (b), shows the result of moving the costs from f to g. Observe that costs of tuples t such that $var(t) = \{x_i, x_j, x_k\}$ are preserved.

4 Mini Buckets with Propagation

In this Section we introduce a refinement of MBE. It consists on performing a movement of costs in each bucket before processing it. We incorporate the concept of equivalence-preserving transformation into MBE, but only at the bucket level. The idea is to *move* costs between minibuckets aiming at a propagation effect. We pursue the accumulation of as much information as possible in one of the mini-buckets.

The following example illustrates and motivates the idea. Suppose that MBE is processing a bucket containing two functions f and g, each one forming a mini-bucket. Variable x_i is the one to be eliminated. Standard MBE would process independently each minibucket, eliminating variable x_i in each function. It is precisely this independent elimination of x_i from each mini-bucket where the lower bound of MBE may lose accuracy. Ideally (i.e, in BE), f and g should be added and their information should *travel* together along the different buckets. However, in MBE their information is split into two pieces for complexity reasons. What we propose is to transfer costs from f to g (or conversely) before processing the mini-buckets. The purpose is to put as much information as possible in the same mini-bucket, so that all this information is jointly processed as BE would do. Consequently, the pernicious effect of splitting the bucket into mini-buckets will presumably be minimized. Figure 2 depicts a numerical illustration. Consider functions f and g from Figure 2 (a). If variable x_i is eliminated independently, we obtain the functions in Figure 2 (c). If the problem contains no more functions,

function $MBE^p(z)$
1. **for each** $i = n..1$ **do**
2. $\quad \mathcal{B} := \{f \in F \mid x_i \in var(f)\}$;
3. $\quad \{\mathcal{P}_1, \ldots, \mathcal{P}_k\} :=$ Partition$(\mathcal{B}, z)$;
4. $\quad$ **for each** $j = 1..k$ **do** $g_j := \sum_{f \in \mathcal{P}_j} f$;
5. $\quad (V, E) :=$ `PropTree`$(\{g_1, \ldots, g_k\})$;
6. $\quad$ `Propagation`$((V, E))$;
7. $\quad$ **for each** $j = 1..k$ **do** $g_j := g_j \downarrow x_i$;
8. $\quad F := (F \cup \{g_1, \ldots, g_k\}) - \mathcal{B}$;
9. **endfor**
10. **return**(g_1);

endfunction

procedure `Propagation`$((V, E))$
11. **repeat**
12. $\quad$ select a node j s.t it has received the messages from all its children;
13. $\quad h_j := g_j[var(g_j) \cap var(g_{parent(j)})]$;
14. $\quad g_j := g_j - h_j$;
15. $\quad g_{parent(j)} := g_{parent(j)} + h_j$;
16. **until** root has received all messages from its children;

endprocedure

Fig. 3. Mini-Bucket Elimination with Propagation (preliminary version). Given a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a zero-arity function g_1 with a lower bound of the optimum cost.

the final lower bound will be 3. Consider now the functions in Figure 2 (b) where costs have been moved from f to g. If variable x_i is eliminated independently, we obtain the functions in Figure 2 (d), with which the lower bound is 5.

The previous example was limited to two mini-buckets containing one function each. Nevertheless, the idea can be easily generalized to arbitrary mini-bucket arrangements. At each bucket $\mathcal{B}$, we construct a *propagation tree* $T = (V, E)$ where nodes are associated with mini-buckets and edges represent movement of costs along branches from the leaves to the root. Each node waits until receiving costs from all its children. Then, it sends costs to its parent. This flow of costs accumulates and propagates costs towards the root.

The refinement of MBE that incorporates this idea is called MBE^p. In Figure 3 we describe a preliminary version. A more efficient version regarding space will be discussed in the next subsection. MBE^p and MBE are very similar and, in the following, we discuss the main differences. After partitioning the bucket into mini-buckets (line 3), MBE^p computes the sum of all the functions in each mini-bucket (line 4). Next, it constructs a propagation tree $T = (V, E)$ with one node j associated to each function g_j. Then, costs are propagated (lines 6, 11-16). Finally, variable x_i is eliminated from each mini-bucket (line 7) and resulting functions are added to the problem in replacement of the bucket (line 8).

Procedure `Propagation` is also depicted in Figure 3. Let j be an arbitrary node of the propagation tree such that has received costs from all its children. It must send costs to its parent $parent(j)$. First, it computes in function h_j the costs that can be sent from j to its parent (line 13). Then, function h_j is subtracted from g_j and summed to $g_{parent(j)}$ (lines 14 and 15). The propagation phase terminates when the root receives costs from all its children.

4.1 Improving the Space complexity

Observe that the previous implementation of MBE^p (Figure 3) computes in two steps (lines 4 and 7), what plain MBE computes in one step. Consequently, MBE^p stores functions with arity up to $z+1$ while MBE only stores functions with arity up to z. Therefore, the previous description of MBE^p has a space complexity slightly higher than MBE, given the same value of z. In the following, we show how the complexity of MBE^p can be made similar to the complexity of MBE. First, we extend the concept of movement of costs to deal with sets of functions. Let F and G be two sets of costs functions. Let $var(F) = \cup_{f\in F} var(f)$, $var(G) = \cup_{g\in G} var(g)$ and $Y = var(F) \cap var(G)$. The *movement of costs* from F to G is done sequentially in three steps:

$h := (\sum_{f\in F} f)[Y]$
$F := F \cup \{-h\}$
$G := G \cup \{h\}$

where $-h$ means that costs contained in h are to be subtracted instead of summed, when evaluating costs of tuples on F. Observe that the first step can be efficiently implemented as,

$$\forall_{t\in Y}, h(t) := \min_{(t'\in var(F) \texttt{ s.t. } t'=t\cdot t'')}\{\sum_{f\in F} f(t')\}$$

This implementation avoids computing the sum of all the functions in F. The time complexity of the operation is $O(d^{|var(F)|})$. The space complexity is $O(d^{|Y|})$.

Figure 4 depicts the new version of MBE^p. The difference with the previous version is that functions in mini-buckets do not need to be summed before the propagation phase (line 4 is omitted). Procedure `Propagation` moves costs between mini-buckets preserving the set of original functions. Line 7, sums the functions in the mini-buckets and eliminates variable x_i in one step, as plain MBE would do.

Observe that the time complexity of line 13 is $O(d^{z+1})$, because $|var(\mathcal{P}_j)| \leq z+1$ (by definition of mini-bucket). The space complexity is $O(d^z)$ because $|var(h)| \leq z$ (note that $var(\mathcal{P}_j) \neq var(\mathcal{P}_{parent(j)})$ because otherwise they would have been merged into one mini-bucket). The previous observation leads to the following result.

Theorem 1. *The time and space complexity of MBE^p is $O(d^{z+1})$ and $O(d^z)$, respectively, where d is the largest domain size and z is the value of the control parameter.*

function $MBE^p(z)$
for each $i = n..1$ **do**
  $\mathcal{B} := \{f \in F |\ x_i \in var(f)\}$;
  $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$:= Partition($\mathcal{B}, z$);
  (V, E) := PropTree($\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$);
  Propagation((V, E));
  for each $j = 1..k$ **do** $g_j := ((\sum_{f \in \mathcal{P}_j} f) - h_j) \downarrow x_i$;
  $F := (F \cup \{g_1, \ldots, g_k\}) - \mathcal{B}$;
endfor
return(g_1);
endfunction
procedure `Propagation`((V, E))
repeat
  select a node j s.t it has received the messages from all its children;
  $h_j := (\sum_{f \in \mathcal{P}_j} f)[var(\mathcal{P}_j) \cap var(\mathcal{P}_{parent(j)})]$;
  $\mathcal{P}_j := \mathcal{P}_j \cup \{-h_j\}$;
  $\mathcal{P}_{parent(j)} := \mathcal{P}_{parent(j)} \cup \{h_j\}$;
until root has received all messages from its children;
endprocedure

Fig. 4. Mini-Bucket Elimination with Propagation. Given a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F})$, the algorithm returns a zero-arity function g_1 with a lower bound of the optimum cost.

4.2 Computation of the Propagation Tree

In our preliminary experiments we observed that the success of the propagation phase of MBE^p greatly depends on the flow of information, which is captured in the propagation tree. In the following we discuss two ideas that heuristically lead to good propagation trees. Then, we will propose a simple method to construct good propagation trees.

For the first observation, consider MBE with $z = 1$ in a problem with four binary functions $f_1(x_1, x_2), f_2(x_2, x_3), f_3(x_2, x_4), f_4(x_3, x_4)$. Variable x_4 is the first to be eliminated. Its bucket contains f_3 and f_4. Each function forms a mini-bucket. MBE^p must decide whether to move costs from f_3 to f_4 or conversely. Observe that after the elimination of x_4, f_4 will go to the bucket of x_3 where it will be summed with f_2. Then, they will go to the bucket of x_2. However, f_3 will *jump* directly to the bucket of x_2. For this reason, it seems more appropriate to move costs from f_3 to f_4. In f_4 the costs go to a higher mini-bucket, so they have more chances to propagate useful information. One way to formalize this observation is the following: We associate to each mini-bucket $\mathcal{P}_j$ a binary number $N_j = b_n b_{n-1} \ldots b_1$ where $b_i = 1$ iff $x_i \in \mathcal{P}_j$. We say that mini-bucket $\mathcal{P}_j$ is smaller than $\mathcal{P}_k$ (noted $\mathcal{P}_j < \mathcal{P}_k$) if $N_j < N_k$. In our propagation trees parents will always be larger than their children.

For the second observation, consider three functions $f(x_7, x_6, x_5, x_4)$, $g(x_7, x_3, x_2, x_1)$, $h(x_7, x_6, x_5, x_1)$. Observe that f shares 1 variable with g and 3 with h. The number of common variables determines the arity of the function that is used as a *bridge* in the cost transfer. The narrower the bridge, the less information that can be captured. Therefore, it seems better to move costs between f and h than between f and g.

In accordance with the two previous observations, we construct the propagation tree as follows: the parent of mini-bucket $\mathcal{P}_u$ will be mini-bucket $\mathcal{P}_w$ such that $\mathcal{P}_u < \mathcal{P}_w$ and they share a maximum number of variables. This strategy combines the two criteria discussed above.

5 Experimental Results

We have tested our approach in three different domains. The purpose of the experiments is to evaluate the effectiveness of the propagation phase and the impact of the propagation tree on that propagation. To that end, we compare the lower bound obtained with three algorithms: standard MBE, MBE with bucket propagation using as a propagation tree a chain of mini-buckets randomly ordered (i.e., MBE^p_r), and MBE with bucket propagation using a propagation tree heuristically built as explained in Section 4.2 (i.e., MBE^p_h). For each domain, we execute those three algorithms with different values of the control parameter z in order to analyze its effect (the highest value of z reported is the highest feasible value given the available memory). In all our experiments, the order of variable elimination is established with the *min-fill* heuristic. All the experiments are executed in a Pentium IV running Linux with 2Gb of memory and 3 GHz.

5.1 Scheduling

For our first experiment, we consider the scheduling of an earth observation satellite. We experiment with instances from Spot5 satellite [15]. These instances have unary, binary and ternary cost functions, and domains of size 2 and 4. Some instances include in their original formulation an additional capacity constraint that we discard on this benchmark.

Figure 5 shows the results. The first column identifies the instance. The second column indicates the value of the control parameter z with which the algorithms are executed. Columns third and fourth report the lower bound obtained and the execution time for standard MBE, respectively. Columns fifth and sixth indicates for MBE^p_r the percentage increment of the lower bound measured as $((Lb_{MBE^p_r} - Lb_{MBE})/Lb_{MBE}) * 100$ and the execution time. Columns seventh and eighth reports the same information for MBE^p_h.

The first thing to be observed is that the results obtained with MBE^p_r does not follow a clear tendency. MBE^p_r increases and decreases the lower bound obtained with standard MBE almost the same times. However, MBE^p_h increases the lower bound obtained with MBE for all the instances. Moreover, when both MBE^p_r and MBE^p_h increase the lower bound, MBE^p_h is always clearly superior. Therefore, it is clear that an adequate propagation tree impacts on the bounds obtained.

Instance	z	MBE(z)		MBE_r^p(z)		MBE_h^p(z)	
		Lb.	Time(sec.)	%	Time(sec.)	%	Time(sec.)
1506	20	184247	827.63	1.6	1628.93	29.8	1706.6
	15	163301	25.43	-5.5	51.48	30.6	51.39
	10	153274	1.33	-13.7	2.65	21.5	2.64
1401	20	184084	691.08	16.8	1469.36	58.6	1574.26
	15	170082	20.82	4.7	47.35	45.8	46.92
	10	155075	1.02	-10.3	2.13	53.5	2.17
1403	20	181184	814.55	7.1	1702.82	59.6	1919.48
	15	162170	27.82	7.3	55.94	57.3	56.9
	10	146155	1.3	10.9	2.58	60.2	2.6
1405	20	191258	1197.06	0.5	2537.64	42.3	2622.88
	15	169233	33.88	-2.3	93.88	54.9	81.17
	10	142206	1.7	-25.3	3.51	64.7	3.5
1407	20	191342	1415.91	-4.0	2935.78	53.8	3008.78
	15	166298	47.44	3.5	94.17	60.1	102.78
	10	144264	2.03	13.8	4.19	68.6	4.23
28	20	134105	252.14	2.2	500.97	38.0	510.72
	15	121105	7.77	-1.6	15	52.8	16.16
	10	103105	0.36	16.4	0.71	49.4	0.71
29	20	8058	4.92	-0.01	5.3	0.01	5.32
	15	8055	0.28	-0.1	0.34	0.02	0.34
	10	8050	0.01	-0.01	0.02	0.07	0.02
408	20	5212	51.19	19.1	75.39	19.3	72.5
	15	5200	2.11	18.7	3.29	19.3	3.41
	10	2166	0.11	38.1	0.2	139.0	0.2
412	20	17314	167.91	5.4	278.29	40.5	278.7
	15	15270	6.49	6.2	10.98	72.1	11.1
	10	10233	0.27	87.8	0.5	88.4	0.78
414	20	23292	629.36	-12.9	1278.39	17.4	1306.98
	15	18268	20.14	-16.3	42.87	49.4	42.99
	10	16213	1.05	-31.0	2.35	49.8	2.09
42	20	127050	38.9	-4.7	71.47	7.8	68.35
	15	111050	1.43	-1.8	2.52	14.4	2.55
	10	93050	0.06	2.1	0.12	19.3	0.12
505	20	19240	51.36	-36.3	66.9	5.2	63.16
	15	16208	2.2	-18.5	3.35	0.1	3.23
	10	13194	0.15	-15.2	0.21	15.1	0.21
507	20	16292	276.74	-6.1	510.66	0.2	520.3
	15	14270	9.84	6.7	19.01	42.2	18.88
	10	11226	0.47	8.6	0.92	53.7	0.92
509	20	22281	507.64	4.6	1026.43	22.5	1046.89
	15	20267	16.2	-24.6	34.68	34.7	34.72
	10	14219	0.83	14.0	1.64	77.7	1.62

Fig. 5. Experimental results on Spot5 instances.

Regarding MBE_h^p, it increases up to 139% the lower bound with respect MBE (e.g. instance 408). The mean increment is 54%, 38%, and 28% when the value of the control parameter z is 10, 15, and 20, respectively. Note that the effect of the propagation is higher for lower values of z because, as we increase the value of z, the number of functions in each mini-bucket increases and the number of mini-buckets decreases. Therefore, the propagated information also decreases and the effect of the propagation is diminished. Moreover, the lower bounds obtained with MBE_h^p and z set to 10 outperforms the ones obtained with MBE and z set to 20 in almost all the instances, which means that the time and space required for obtaining a bound of a given quality is decreased.

Regarding cpu time, MBE_h^p is from 2 to 3 times slower than MBE. The reason is that cost functions are evaluated twice: the first one during the propagation

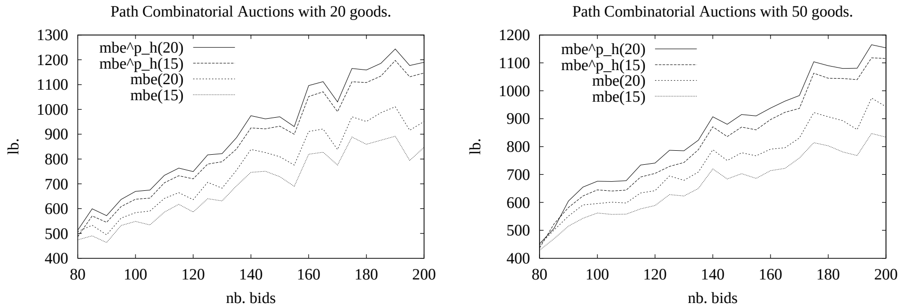

Fig. 6. Combinatorial Auctions. Path distribution.

phase for establishing the costs to be moved, and the second one during the regular process of variable elimination. However, it is important to note that it is the space and not the time what bounds the maximum value of z that can be used in practice. As a consequence, that constant increase in time is not that significant as the space complexity remains the same.

5.2 Combinatorial Auctions

Combinatorial auctions (CA) allow bidders to bid for indivisible subsets of goods. Consider a set of goods $\{1, 2, \ldots, n\}$ that go on auction. There are m bids. Bid j is defined by the subset of requested goods $X_j \subseteq \{1, 2, \ldots, n\}$ and the money offer b_j. The bid-taker must decide which bids are to be accepted maximizing the benefits.

We have generated CA using the *path* and *regions* model of the CATS generator [16]. We experiment on instances with 20 and 50 goods, varying the number of bids from 80 to 200. For each parameter configuration, we generate samples of size 10. We execute algorithms MBE, MBE_r^p, and MBE_h^p with z equal to 15 and 20. We do not report results with MBE_r^p because it was always very inferior than MBE_h^p. For space reasons, we only report results on the *path* model. The results for the *regions* model follows the same pattern.

Figure 6 reports the results for *path* instances with 20 and 50 goods, respectively. As can be observed, the behaviour for both configurations is almost the same. Regarding the algorithms, it is clear that MBE_h^p always outperformes MBE. Note that the lower bound obtained with $MBE_h^p(z = 15)$ is clearly superior than that obtained with $MBE(z = 20)$. Moreover, as pointed out in the previous domain, the effect of the propagation in each sample point is higher for $z = 15$ than for $z = 20$. That is, the percentage of increment in the lower bound obtained with $MBE_h^p(z = 15)$ is higher than that of $MBE_h^p(z = 20)$. Finally, it is important to note that the impact of the propagation is higher when the problems become harder (i.e., as the number of bids increase).

5.3 Maxclique

A *clique* of a graph $G = (V, E)$ is a set $S \subseteq V$, such that every two nodes in S are joined by an edge of E. The *maximum clique problem* consists on finding the largest cardinality of a clique. The maximum clique problem can be easily encoded as a minimization problem (i.e., minimize the number of nodes in $V - S$).

We test our approach on the dimacs benchmark [17]. Figure 7 reports the results. The first column identifies the instance. The second column indicates the value of the control parameter z with which the algorithms are executed. The third column report the lower bound obtained with standard MBE. Columns fourth and fifth indicates, for MBE^p_r and MBE^p_l, the percentage of increment in the lower bound with respect MBE, respectively. As the behaviour of the cpu time is the same as for the previous benchmark, we do not report this information.

MBE^p_r increases the lower bound obtained with standard MBE for all the instances except for those of *hamming* and *johnson*. The percentage of increment is up to 1226% when the value of the control parameter z is 10, and up to 812% when z is the highest value. The best results are obtained with MBE^p_h which obtains a percentage increment of 1566% (see instance *p-hat1500-2*). In this case, the increase ranges from 14.6% to 1566% when z is set to 10, and from 17.6% to 1292% for the highest value of z.

It is important to note that the bound obtained with MBE^p_h is always higher than that of MBE^p_r. For some instances, the percentage of increment of MBE^p_h is more than 4 times higher the one obtained with MBE^p_r (e.g. instance *c-fat200-1*). Therefore, it is clear that an adequate propagation tree impacts on the propagation phase and, as a consequence, on the bounds obtained.

6 Conclusions and Future Work

Mini-bucket elimination (MBE) is a well-known approximation algorithm for combinatorial optimization problems. It has a control parameter z which allow us to trace time and space for approximation accuracy. In practice, it is usually the space rather than the cpu time which limits the control parameter.

In this paper we introduce a new propagation phase that MBE should execute at each bucket. In the new algorithm, that we call MBE^p, the idea is to *move* costs along mini-buckets in order to accumulate as much information as possible in one of them. The propagation phase is based on a *propagation tree* where each node is a mini-bucket and edges represent movements of costs along branches from the leaves to the root. Finally, it is important to note that the propagation phase does not increase the asymptotical time and space complexity of the original MBE algorithm.

We demonstrate the effectiveness of our algorithm in *scheduling*, *combinatorial auction* and *maxclique* problems. The typical percentage of increment in the lower bound obtained is 50%. However, for almost all maxclique instances the percentage of increment ranges from 250% to a maximum of 1566%. Therefore,

MBE^p is able to obtain much more accurate lower bounds than standard MBE using the same amount of resources.

In our future work we want to integrate the propagation phase into the depth-first mini-bucket elimination algorithm [9]. The two main issues are how the computation tree rearrangements affect the bucket propagation and how to efficiently deal with the functions maintaining the transferred costs.

Acknowledgement

This research has been funded with project TIN2005-09312-C03-02.

References

1. Dechter, R., Rish, I.: Mini-buckets: A general scheme for bounded inference. Journal of the ACM **50** (2003) 107–153
2. 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
3. Pearl, J.: Probabilistic Inference in Intelligent Systems. Networks of Plausible Inference. Morgan Kaufmann, San Mateo, CA (1988)
4. H. Xu, R.R., Sakallah, K.: sub-sat: A formulation for relaxed boolean satisfiability with applications in rounting. In: Proc. Int. Symp. on Physical Design, CA (2002)
5. D.M. Strickland, E.B., Sokol, J.: Optimal protein structure alignment using maximum cliques. Operations Research **53** (2005) 389–402
6. Vasquez, M., Hao, J.: A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. Journal of Computational Optimization and Applications **20(2)** (2001)
7. Park, J.D.: Using weighted max-sat engines to solve mpe. In: Proc. of the 18^{th} AAAI, Edmonton, Alberta, Canada (2002) 682–687
8. Kask, K., Dechter, R.: A general scheme for automatic generation of search heuristics from specification dependencies. Artificial Intelligence **129** (2001) 91–131
9. Rollon, E., Larrosa, J.: Depth-first mini-bucket elimination. In: Proc. of the 11^{th} CP, Sitges (Spain), LNCS 3709. Springer-Verlag. (2005) 563–577
10. Cooper, M., Schiex, T.: Arc consistency for soft constraints. Artificial Intelligence **154** (2004) 199–227
11. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113** (1999) 41–85
12. Bertele, U., Brioschi, F.: Nonserial Dynamic Programming. Academic Press (1972)
13. Larrosa, J., Schiex, T.: Solving weighted csp by maintaining arc-consistency. Artificial Intelligence **159** (2004) 1–26
14. Cooper, M.: High-order consistency in valued constraint satisfaction. Constraints **10** (2005) 283–305
15. Bensana, E., Lemaitre, M., Verfaillie, G.: Earth observation satellite management. Constraints **4(3)** (1999) 293–299
16. K.Leuton-Brown, M., Y.Shoham: Towards a universal test suite for combinatorial auction algorithms. ACM E-Commerce (2000) 66–76
17. Johnson, D.S., Trick, M.: Second dimacs implementation challenge: cliques, coloring and satisfiability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS **26** (1996)

Instance	z	MBE	MBE_r^p	MBE_h^p
		Lb.	%	%
brock200-1	18	66	30.3	48.4
	10	51	52.9	78.4
brock200-2	18	55	67.2	103.6
	10	29	200	268.9
brock200-3	18	64	48.4	68.7
	10	38	139.4	173.6
brock200-4	18	63	36.5	65.0
	10	41	121.9	131.7
brock400-1	18	79	100	141.7
	10	46	256.5	273.9
brock400-2	18	75	114.6	157.3
	10	44	261.3	277.2
brock400-3	18	87	88.5	114.9
	10	44	250	286.3
brock400-4	18	76	106.5	160.5
	10	47	248.9	289.3
brock800-1	18	71	336.6	454.9
	10	41	675.6	773.1
brock800-2	18	63	395.2	520.6
	10	37	748.6	875.6
brock800-3	18	68	352.9	483.8
	10	44	604.5	706.8
brock800-4	18	71	343.6	460.5
	10	36	758.3	902.7
c-fat200-1	18	71	32.3	78.8
	10	62	27.4	112.9
c-fat200-2	18	63	38.0	82.5
	10	48	77.0	156.2
c-fat200-5	18	55	23.6	12.7
	10	37	32.4	70.2
c-fat500-10	18	77	115.5	123.3
	10	52	173.0	253.8
c-fat500-1	18	132	84.0	137.1
	10	107	126.1	196.2
c-fat500-2	18	108	108.3	164.8
	10	85	160	254.1
c-fat500-5	18	83	145.7	202.4
	10	74	163.5	264.8
hamming10-2	18	412	-66.9	-72.0
	10	419	-72.0	-73.7
hamming10-4	18	119	264.7	413.4
	10	77	451.9	720.7
hamming6-2	18	32	-28.1	-31.2
	10	32	-50	-59.3
hamming6-4	18	45	-4.4	2.2
	10	33	9.0	33.3
hamming8-2	18	114	-59.6	-64.9
	10	113	-74.3	-78.7
hamming8-4	18	82	46.3	89.0
	10	51	113.7	215.6
johnson16-2-4	18	72	-4.1	11.1
	10	56	10.7	48.2
johnson32-2-4	18	195	27.6	71.2
	10	134	75.3	150
johnson8-2-4	18	23	-4.3	0
	10	20	-20	-5
johnson8-4-4	18	45	-22.2	-11.1
	10	40	-15	-10
keller4	18	70	27.1	54.2
	10	41	97.5	168.2
keller5	18	90	246.6	394.4
	10	61	414.7	634.4
MANN-a27	15	247	0.4	0.4
	10	244	-1.2	0.8

Instance	z	MBE	MBE_r^p	MBE_h^p
		Lb.	%	%
MANN-a45	15	677	-0.7	0.4
	10	671	-0.1	0.1
MANN-a81	15	2177	0.0	0.3
	10	2171	-0.1	0.5
p-hat1000-1	15	85	380	654.1
	10	63	577.7	873.0
p-hat1000-2	15	57	589.4	821.0
	10	36	1013.8	1325
p-hat1000-3	15	82	364.6	415.8
	10	50	668	764
p-hat1500-1	15	69	802.8	1292.7
	10	82	686.5	1021.9
p-hat1500-2	15	64	812.5	1112.5
	10	45	1226.6	1566.6
p-hat1500-3	15	79	624.0	706.3
	10	54	924.0	1111.1
p-hat300-1	18	62	112.9	195.1
	10	48	187.5	306.2
p-hat300-2	18	61	121.3	168.8
	10	38	247.3	328.9
p-hat300-3	18	76	71.0	100
	10	51	145.0	172.5
p-hat500-1	18	74	170.2	301.3
	10	50	330	524
p-hat500-2	18	75	178.6	248
	10	39	407.6	556.4
p-hat500-3	18	93	125.8	169.8
	10	50	300	338
p-hat700-1	15	66	340.9	581.8
	10	52	482.6	711.5
p-hat700-2	18	63	357.1	492.0
	10	36	672.2	919.4
p-hat700-3	18	78	260.2	330.7
	10	44	543.1	588.6
san1000	15	89	319.1	493.2
	10	100	260	438
san200-0.7-1	18	69	26.0	53.6
	10	50	82	86
san200-0.7-2	18	84	40.4	51.1
	10	53	75.4	115.0
san200-0.9-1	18	108	-1.8	0
	10	82	18.2	14.6
san200-0.9-2	18	85	20	17.6
	10	68	25	27.9
san200-0.9-3	18	83	21.6	18.0
	10	67	34.3	26.8
san400-0.5-1	18	79	115.1	194.9
	10	58	189.6	289.6
san400-0.7-1	18	84	95.2	144.0
	10	55	138.1	209.0
san400-0.7-2	18	78	105.1	158.9
	10	42	247.6	309.5
san400-0.7-3	18	73	138.3	180.8
	10	47	225.5	287.2
san400-0.9-1	18	97	63.9	75.2
	10	75	93.3	98.6
sanr200-0.7	18	61	42.6	63.9
	10	45	80	104.4
sanr200-0.9	18	77	12.9	23.3
	10	61	31.1	37.7
sanr400-0.5	18	67	152.2	223.8
	10	32	406.2	543.7
sanr400-0.7	18	76	103.9	152.6
	10	47	231.9	270.2

Fig. 7. Experimental results on maxclique instances.