Bucket Elimination for Multiobjective Optimization Problems and its Application to Global Constraints

Javier Larrosa, Emma Rollón

Universitat Politecnica de Catalunya
Barcelona, Spain
{larrosa, erollon}@lsi.upc.edu

Abstract. Multiobjective optimization deals with problems involving multiple measures of performance that should be optimized simultaneously. In this paper we extend bucket elimination (BE), a well known dynamic programming generic algorithm, from mono-objective to multiobjetive optimization. We show that the resulting algorithm, MO-BE, can be applied to true multi-objective problems as well as mono-objective problems with knapsack (or related) global constraints. We also extend mini-bucket elimination (MBE), the approximation form of BE, to multiobjective optimization. The new algorithm MO-MBE can be used to obtain good quality *multi-objective* lower bounds or it can be integrated into multi-objective branch and bound in order to increase its pruning efficiency. Its accuracy is empirically evaluated in real scheduling problems, as well as in the MaxSAT-MaxOnes problem.

1 Introduction

In constraint satisfaction problems (CSPs) the task is to find values for a set of variables such that a set of constraints is satisfied [?]. Such satisfying assignments are the problem solutions. In *soft* CSPs the task is to find the best solution according to some preferences expressed by means of cost functions [?]. Given their optimization nature, soft CSPs are also known as *constraint optimization problems*. Without loss of generality, in the following we will assume optimization as minimization. Algorithms for CSPs and soft CSPs can be divided into *exact* and *approximate*. Exact algorithms are normally based on search (e.g. branch and bound, BB [?]) or dynamic programming (e.g., bucket elimination, BE [?]). Approximate algorithm can compute upper bounds (e.g., metaheuristics [?]) or lower bounds (e.g. relaxation methods [?]). Algorithms that compute lower bounds (such as mini-bucket elimination, MBE [?]) are a fundamental component of branch and bound because they can be executed at every search node in order to detect infeasible nodes [?].

Many real world problems involve multiple measures of performance, or objectives, which should be optimized simultaneously. The simultaneous optimization of multiple, possibly competing, objective functions deviates from single function optimization in that it seldom admits a single, perfect solution. Instead, multiobjective constraint optimization problems tend to be characterized by a family of alternatives which must be considered equivalent in the absence of information concerning the relevance of each objective relative to the others.

Algorithms for multiobjective optimization can also be divided into exact and approximate. During our bibliographic research (see [**?**]) we observed that most work has been done in approximate algorithms to compute upper bounds. In particular, evolutionary approaches seem to be the best option in practice. Very little is known about approximate algorithms to compute lower bounds. Regarding exact algorithms, most work focus on specific problems such as the multiobjective version of: knapsack problem, traveling salesman problem, shortest path problem, *etc*.

In this paper we consider generic algorithms for multiobjective optimization. In particular, we focus on the extension to multiobjective optimization of *bucket elimination* (BE), a well known algorithm based on dynamic programming. First, we define the *multiobjective weighted constraint satisfaction* framework (MO-WCSP), where all their objective functions are additive. Then, we extend BE from WCSP to MO-WCSP, producing MO-BE. Next, we show that MO-BE can be used not only to solve MO-WCSP problems, but also to deal efficiently with knapsack-like constraints. Moreover, we show that this approach provides the same complexity improvement as our previous work [**?**,**?**] while avoiding the introduction of state-variables and other *ad-hoc* techniques.

We also show how *mini-buckets* (MBE), the approximation version of BE, can be extended to multiobjective problems. The new algorithm MO-MBE produces a set of lower bounds of the exact solutions. The relevance of this algorithm is two-fold: on the one hand it can be used to bound the optimum of a problem when computing the exact solutions is too difficult. On the other hand, it can be used in combination with branch and bound to enhance its pruning power. The efficiency and accuracy of MO-MBE is empirically demonstrated in randomly generated problems with two objective functions.

The structure of this paper is as follows: Section 2 provides some preliminaries on mono-objective optimization, Section 3 introduces multiobjective WCSPs, Section 4 introduces the extension of BE to multiobjective optimization, Section 5 shows how some global constraints in mono-objective optimization can be processed more efficiently as multiobjective, Section 6 introduces multiobjective mini-buckets, Section 7 reports some experimental results and, finally Section 8 gives some conclusions and points out some directions of future work.

2 Preliminaries

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. We call d the largest domain size. 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. When the scope is clear by the context, we omit the variables and express the tuple as a sequence of domain values $(a_{i_1} \dots a_{i_k})$. 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 *constraint satisfaction problem* (CSP) is a triplet $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{R})$, where $\mathcal{R}$ is a set of constraints defining the variables' simultaneous legal value assignments. We say that a tuple t *satisfies* a constraint R if $var(R) \subseteq var(t)$ and $t[var(R)] \in R$. A *solution* of a CSP is a complete instantiation X that satisfies all the constraints in $\mathcal{R}$. In a CSP, the usual task of interest is to find a solution or prove that there is none. Solving a CSP is in general NP-complete.

Example 1. Consider a problem with four objects that we must either take or leave behind. We can represent this by four variables (x_1, x_2, x_3, x_4) and two values per domain $D_i = \{0, 1\}$ (1 and 0 mean *take* and *discard*, respectively). Suppose the existence of three constraints that must be satisfied: $x_3 \vee x_4$, $\overline{x_4 \wedge x_2}$, and $x_1 \veebar x_3$ ($\veebar$ means *exclusive or*). The problem is a CSP with four solutions: 0010, 0011, 0110 and 1001.

Weighted CSP (WCSP) [**?**,**?**] are CSPs where the set of constraints is replaced by a set of cost functions ($\mathcal{F}$) which denote preferences among tuples. A *cost function* f over $S \subseteq \mathcal{X}$ associates costs to tuples t such that $var(t) = S$. The set of variables S is the *scope* of f and is noted $var(f)$. Abusing notation, when $var(f) \subset var(t)$, $f(t)$ will mean $f(t[var(f)])$. In the sequel, we assume costs to be natural numbers. The set $\mathcal{F}$ defines the objective function,

$$F(X) = \sum_{f \in \mathcal{F}} f(X)$$

A WCSP is a quadruple $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F}, K)$, where $\mathcal{X}$ and $\mathcal{D}$ are variables and domains. $\mathcal{F}$ is the set of cost functions. Value K bounds the maximum acceptable cost of solutions. A WCSP *solution* is a complete assignment X such that $F(X) < K$. The task of interest consists on finding a solution with minimum cost, if there is any. Observe that the existence of K is assumed without loss of generality, since it can be set to an arbitrarily large value. However, K can be used to tighten the notion of solution if we want to guarantee a certain degree of quality. Its importance will become clear in Section 5.

Example 2. Consider the problem of Example 1. Suppose that taking object i brings a profit $p_i = i$. Besides, objects 2 and 3 are complementary, meaning that if both of them are taken we get an additional profit $p_{23} = 3$. Making the most profitable selection of objects can be expressed as a minimization WCSP, where the task is to minimize the profit of discarded objects. Hard constraints are expressed as $0, \infty$ functions,

$$h_1(x_1, x_3) = \begin{cases} 0, & x_1 \underline{\vee} x_3 \\ \infty, & otherwise \end{cases}$$

$$h_2(x_3, x_4) = \begin{cases} 0, & x_3 \vee x_4 \\ \infty, & otherwise \end{cases}$$

$$h_3(x_2, x_4) = \begin{cases} 0, & \overline{x_2 \wedge x_4} \\ \infty, & otherwise \end{cases}$$

Soft constraints encoded as unary and binary functions,

$$f_i(x_i) = \begin{cases} i, & x_i = 0 \\ 0, & otherwise \end{cases} \qquad f_{23}(x_2, x_3) = \begin{cases} 0, & x_2 \wedge x_3 \\ 3, & otherwise \end{cases}$$

The value of K can be trivially set to $\sum_{i=1}^{4} f_i(0) + f_{23}(0,0) = 13$, without fear to miss any solution. The optimal solution is 0110 with cost 5. Observe that, if we set $K = 5$, the problem has no solution.

A WCSP can be represented by its *constraint graph* $G = (V, E)$ (also called *interaction graph*), which contains one node per variable and one arc connecting any pair of nodes whose variables are included in the scope of some function. Let o be an ordering of V. The width of node i subject to o, noted $w(o, i)$ is the number of adjacents to i which are before i in o. The *width* of G subject to o, noted $w(o)$, is the maximum width among the nodes. The *induced graph* $G^*(o)$ is computed as follows: Nodes are processed in reverse order, according to o. When processing node i, edges are added as needed in order to make a clique with all its adjacent which are still unprocessed. The width of the induced graph is called the *induced width* and is noted $w^*(o)$ [**?**].

2.1 Bucket Elimination

Bucket elimination (BE) [**?**,**?**] is a generic algorithm that can be used for WCSP solving. It uses two 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 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 to 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.

Unfortunately, in general, the result of summing functions or eliminating variables cannot be expressed intensionally by algebraic expressions. Therefore, we assume functions to be extensionally stored in tables. Thus, the space complexity of storing function f is $O(d^{|var(f)|})$.

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

endfunction

Fig. 1. Bucket Elimination. Given a WCSP $(\mathcal{X}, \mathcal{D}, \mathcal{F}, K)$, the algorithm returns a constant function g_1 (i.e, $var(g_1) = \emptyset$) with the optimal cost, along with one optimal assignment t. If there is no solution, the algorithms returns NIL.

BE (Figure 1) uses an arbitrary variable ordering o that we assume, without loss of generality, lexicographical (i.e, $o = (x_1, x_2, \ldots, x_n)$). BE works in two phases. In the first phase (lines 1-6), 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 5). 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 7-11) 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 7), 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 9,10). We use argmin to denote the argument producing the minimum valuation. The g_i-*subproblem* is the subproblem formed by all the original cost functions involved in the computation of g_i. Let t be an assignment of variables $x_1, \ldots, x_{i-1}$. The correctness of BE is a direct consequence of the fact that when processing bucket $\mathcal{B}_i$, $g_i(t[var(g_i)])$ is the cost of the best extension of t to variables $x_i, x_{i+1}, \ldots, x_n$ in the g_i-subproblem.

Theorem 1. *[?] The complexity of* BE *along ordering o is time $O(e \times d^{w^*(o)+1})$ and space $O(n \times d^{w^*(o)})$, where e is the number of functions, d is the largest domain size, n is the number of variables and $w^*(o)$ is the induced width under the corresponding variable ordering.*

2.2 A non-standard implementation of Bucket Elimination

In this subsection we provide a non-standard implementation of the second phase of the BE algorithm (Figure 1, lines 7-11). Although it may look unnecessarily complex for BE, it will facilitate the comprehension of the new algorithm MO-BE introduced in Section 4. The idea is to retrieve the optimal solution by keeping track of the optimal cost of the different subproblems contained in each bucket.

Let $\mathcal{B}_i = \{f_{i_1}, \ldots, f_{i_{m_i}}\}$ be the set of cost functions of bucket $\mathcal{B}_i$. Each cost function f_{i_k} is either an original function or the result of processing a higher bucket $\mathcal{B}_j$ (i.e., $f_{i_k} = g_j$). We define $db(f_{i_k})$ as the *departure bucket* for function f_{i_k}, that is, the bucket where the function was generated. Therefore, $db(f_{i_k}) = i$ if f_{i_k} is an original function, and $db(f_{i_k}) = j$ if $f_{i_k} = g_j$.

7. $t := \lambda$;
8. $C[1] := g_1$;
9. **for each** $i = 1..n$ **do**
10.   let $\mathcal{B}_i = \{f_{i_1}, f_{i_2}, \ldots, f_{i_{m_i}}\}$
11. $b := pop(\{a \in D_i | \; (\sum_k^{m_i} f_{i_k})(t \cdot (x_i \leftarrow a)) = C[i]\})$;
12. $t := t \cdot (x_i \leftarrow b)$;
13. $(v_1, \ldots, v_{m_i}) := (f_{i_1}(t \cdot (x_i \leftarrow b)), \ldots, f_{i_{m_i}}(t \cdot (x_i \leftarrow b)))$;
14. $\forall 1 \leq k \leq m_i$ **if** $db(f_{i_k}) \neq i$ **then** $C[db(f_{i_k})] := v_k$;
15. **endfor**
16. **return**(g_1, t);

Fig. 2. Second phase of the Bucket Elimination with a non-standard implementation.

As in standard BE, the new second phase of the algorithm (Figure 2) generates in t an optimal assignment of variables, considering them one at the time, from first to last. We use an array $C[1 \ldots n]$. Each $C[i]$ will store the cost contribution of g_i to the solution (namely, the optimal contribution of the g_i-subproblem) . Initially, t is an empty assignment λ (line 7). Clearly, $C[1]$ is set to g_1 (line 8). The optimal value for x_1 is any domain value $b \in \mathcal{D}_1$ such that

$C[1] = \sum_{k=1}^{m_i} f_{1_k}(t \cdot (x_1 \leftarrow b))$. In line 11 one such value is selected and in line 12 added to t. The contribution of each function $f_{1_k} \in \mathcal{B}_1$ to the cost $C[1]$ is $f_{1_k}(t \cdot (x_1 \leftarrow b)$. Therefore, each contribution $f_{1_k}(t \cdot (x_1 \leftarrow b)$ is propagated to the C entry of the corresponding departure bucket $C[db(f_{1_k})]$ (lines 13 and 14). The same procedure is repeated for each variable x_i in increasing order.

2.3 Mini-Bucket Elimination

Mini-Bucket Elimination (MBE) [**?**] is an approximation designed to avoid the space and time problem of full bucket elimination by partitioning large buckets into smaller subsets called mini-buckets which are processed independently. Consider a bucket $\mathcal{B}_i$. The mini-bucket algorithm creates a partition $Q' = \{Q_1, \ldots, Q_m\}$ of the functions in $\mathcal{B}_i$. The approximation processes each subset separately, thus computing a set of functions $\{g_{il}\}_{l=1}^{m}$,

$$g_{il} = (\sum_{f \in Q_l} f) \downarrow x_i$$

Bucket elimination, on the other hand, would compute the function g_i,

$$g_i = (\sum_{f \in \mathcal{B}_i} f) \downarrow x_i$$

Since,

$$\overbrace{(\sum_{f \in \mathcal{B}_i} f) \downarrow x_i}^{g_i} \geq \sum_{l=1}^{m} \overbrace{(\sum_{f \in Q_l} f) \downarrow x_i}^{g_{il}}$$

the bound computed in each bucket yields a lower bound on the cost of the solution. The quality of the bound depends on the partitioning procedure. Given a bounding parameter k, the algorithm creates a k-partition, where each mini-bucket includes no more than k variables. In general, greater values of k increment the number of functions included in each mini-bucket. Therefore, the bound will be presumably closer to the cost of the optimal solution.

Theorem 2. *[?] The complexity of* MBE(k) *is time $O(e \times d^k)$ and space $O(e \times d^{k-1})$, where e is the number of functions.*

3 Multiojective Weighted Constraint Satisfaction Problem

A *multiobjective* WCSP (MO-WCSP) is defined as $(\mathcal{X}, \mathcal{D}, < \mathcal{F}_i, K_i >_{i=1}^{p})$, where $\mathcal{X} = \{x_1, \ldots, x_n\}$ and $\mathcal{D} = \{D_1, \ldots, D_n\}$ are variables and domains. Each $\mathcal{F}_i$ is a set of cost functions defining an objective function $F_i(X) = \sum_{f \in \mathcal{F}_i} f(X)$. Value K_i bounds the maximum acceptable cost for the objective function F_i. There is a vector-valued objective function $F(X) = (F_1(X), \ldots, F_p(X))$. Given a complete assignment X, we say that $F(X)$ is *consistent* iff $\forall i, F_i(X) < K_i$. A *solution* is a complete assignment X such that $F(X)$ is consistent. Given two solutions X and X' such that $F(X) \neq F(X')$, we say that X is *better* than X' ($F(X) < F(X')$) if $\forall i, F(X) \leq F(X')$. If $F(X) < F(X')$, we say that $F(X)$ *dominates* $F(X')$. A solution X is *efficient* or *pareto optimal* if there is no better solution. Let $\mathcal{X}_E$ be the set of efficient solutions, and Z be their set of non-dominated costs, also called the *efficient frontier*. The task in a MO-WCSP is to compute Z (and, possibly, one or all efficient solutions for each of its elements). Observe that $|Z| = O(\prod_{i=1}^{p-1} K_i)$. However, $\mathcal{X}_E$ can be as large as $O(d^n)$, when every assignment is efficient. Clearly, when there is only one objective function (i.e., $p = 1$), MO-WCSP is equivalent to WCSP, and $|Z| = O(1)$. As in the WCSP case, each MO-WCSP instance has an associated constraint graph G computed in the same way.

Example 3. Consider that we add a weight $w_i = 5 - i$ and a volume $v_i = i$ to every object in our running example. We would like to select valuable objects but, in addition, we do not want to carry a heavy and big container. The original objective function (profit of discarded objects) is now called F_1. We need to define new cost functions F_2 and F_3 for weight and volume. The new problem has three solutions: 0010, 0110, and 0011, with non-dominated costs $(10, 2, 3)$, $(5, 5, 5)$, and $(6, 3, 7)$, respectively.

If objects must be carried in a container with weight and volume bounded by $K_2 = 5$ and $K_3 = 6$, respectively, the problem has only one solution 0010 with cost $(10, 2, 3)$.

4 Bucket Elimination for MO-WCSP

In this Section we extend BE to the MO-WCSP model. The first step is to extend cost functions to the new multiobjective nature. A *multi-cost function* f with scope $var(f) \subseteq \mathcal{X}$ associates to each tuple t in $var(f)$ *a set of non-dominated points in the space of consistent multiobjective costs.* Thus, $f(t) = \{\boldsymbol{v_1}, \ldots, \boldsymbol{v_r}\}$, with $\boldsymbol{v_i} = (v_{i1}, \ldots, v_{ip})$, such that $\forall\ i, j,\ v_{ij} < K_j$ and $\forall\ i, j,\ \boldsymbol{v_i} \not< \boldsymbol{v_j}$. Observe that $|f(t)| = O(\prod_{i=1}^{p-1} K_i)$ and $|f| = O(\prod_{i=1}^{p-1} K_i \times \times d^{|var(f)|})$.

The new algorithm will deal with multi-cost functions. Roughly, the elimination of variable x_i from a multi-cost function will produce a new multi-cost function g_i such that $g_i(t)$ will contain *all the efficient extension of t to the eliminated variables* $x_i, x_{i+1}, \ldots, x_n$, with respect the g_i-subproblem. In the following, $\boldsymbol{v} + \boldsymbol{v}'$ will be the usual component-by-component sum. Operations over multi-cost functions are extended as follows:

– Let f and g be two multi cost functions. Their sum $h = f + g$ is defined as,

$$h(t) = \{\boldsymbol{v}|\ \ t = t' \cdot t'', \boldsymbol{v} = \boldsymbol{v}' + \boldsymbol{v}'', \boldsymbol{v}' \in f(t), \boldsymbol{v}'' \in g(t),$$
$$\texttt{consistent}(\boldsymbol{v}), \texttt{nondominated}(\boldsymbol{v})\}$$

$f + g$ can be trivially computed in time $O(\prod_{i=1}^{p-1} K_i^2 \times d^{|var(f) \cup var(g)|})$.

– Let f be a multi-cost function. The elimination of x_i, $h = f \downarrow x_i$ is defined as,

$$h(t) = \{\boldsymbol{v} | \ \ \boldsymbol{v} \in f(t \cdot (x_i \leftarrow a)), \ \texttt{nondominated}(\boldsymbol{v})\}$$

$h = f \downarrow x_i$ can be computed in $O(\prod_{i=1}^{p-1} K_i \times d^{|var(f)|})$.

Observe that in the $p = 1$ case, these definitions reduce to the classical ones.

function MO-BE$(\mathcal{X}, \mathcal{D}, < \mathcal{F}_i, K_i >_{i=1}^{p})$
  $\mathcal{F} := \emptyset$
  for each $i = 1..p$, $f \in F_i$ **do**
    for each $t \in var(f)$ **do**
      if $f(t) \geq K_i$ **then** $g(t) := \emptyset$
      else $g(t) := \{(v_1, \ldots, v_p)\}$ where $v_i = f(t)$ and $v_j = 0, \forall j \neq i$
    endfor
    $\mathcal{F} := \mathcal{F} \cup \{g\}$
  endfor
  for each $i = n \ldots 1$ **do**
    $\mathcal{B}_i := \{f \in \mathcal{F} \mid x_i \in var(f)\}$;
    $g_i := (\sum_{f \in \mathcal{B}_i} f) \downarrow x_i$;
    if $\forall t, g_i(t) = \emptyset$ **then return** NIL;
    $F := (F \cup \{g_i\}) - \mathcal{B}_i$;
  endfor
  let $g_1 = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_r}\}$
  for each $j = 1..r$ **do**
    $t_j := \lambda$;
    $C[1] := \boldsymbol{v_j}$
    for each $i = 1 \ldots n$ **do**
      let $\mathcal{B}_i = \{f_{i_1}, \ldots, f_{i_{m_i}}\}$
      $\forall a \in D_i \ \ S_a = \{(\boldsymbol{v_1}, \ldots, \boldsymbol{v_{m_i}}) \mid \sum_{k=1}^{m_i} \boldsymbol{v_k} = C[i], \ \forall k, \ \boldsymbol{v_k} \in f_{i_k}(t \cdot (x_i \leftarrow a))\}$
      $b := \text{pop}(\{a \in D_i | \ S_a \neq \emptyset\})$
      $t_j := t_j \cdot (x_i \leftarrow b)$;
      $(\boldsymbol{v_1}, \ldots, \boldsymbol{v_{m_i}}) := \text{pop}(S_b)$
      $\forall 1 \leq k \leq m_i$ **if** $db(f_{i_k}) \neq i$ **then** $C[db(f_{i_k})] := \boldsymbol{v_k}$;
    endfor
  endfor
  return $(g_1 = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_k}\}, \{t_1, t_2, \ldots, t_k\})$;
endfunction

Fig. 3. Description of MO-BE. The input is a MO-WCSP instance $(\mathcal{X}, \mathcal{D}, < \mathcal{F}_i, K_i >_{i=1}^{p})$. The output is g_1, a zero-arity multi-cost function which contains the efficient frontier and, for each element $\boldsymbol{v_j} \in g_1$, an efficient solution t_j. If there is no solution, the algorithm returns NIL.

Figure 3 shows MO-BE, the generalization of BE to MO-WCSP. Its structure is similar to standard BE. In the following, we discuss the main differences. MO-

BE receives a MO-WCSP instance $(\mathcal{X}, \mathcal{D}, < \mathcal{F}_i, K_i >_{i=1}^{p})$. First of all, each cost function $f \in \mathcal{F}_i$ is transformed into a multi-cost function h (lines 1-8). In the new function, each $h(t)$ contains a singleton representing the same information as $f(t)$ but extended to the new vectorial context. This is trivially done as follows: if $f(t) = v$, then $h(t) = \{(v_1, v_2, \ldots, v_p)\}$, where $v_i = v$ and $v_j = 0, \forall j \neq i$. The new multi-cost functions are stored in the set $\mathcal{F}$.

The first phase of the algorithm (lines 9-14) computes the efficient frontier Z. It works as BE, the only difference being that multi cost functions are used instead of standard cost functions. Let $\boldsymbol{v} = (v_1, \ldots, v_p)$ be an element of the multi cost relation $g_i(t)$ computed during the elimination of x_i (i.e, $\boldsymbol{v} \in g_i(t)$). By construction, tuple t can be consistently extended to the eliminated variables $x_i, x_{i+1}, \ldots, x_n$ with cost v_i for each objective function F_i. Besides, such extension is non-dominated. After the first phase, g_1 contains a set of points in the space of solutions, which is exactly the efficient frontier Z of the problem.

Let g_1 contain r vector points $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_r}\}$. The second phase (lines 16-27) computes one efficient solution t_j for every element $\boldsymbol{v_j} \in g_1$. The idea is the same as in the non-standard implementation given in section 2.2, that is, to retrieve the efficient solution keeping track of the cost contribution of each g_i to the solution. In this case, the array $C[i]$ will store a non-dominated multi cost attainable from g_i. Initially, $t_j = \lambda$ and $C[1] = \boldsymbol{v_j}$ is the vector point for which the efficient solution is searched. For each j, variables are considered in increasing order $x_1, \ldots, x_n$ (line 19). The optimal domain value $a \in \mathcal{D}_1$ for x_1 is any one such that $C[1] \in \sum_{k=1}^{m_1} f_{1_k}(t \cdot (x_1 \leftarrow a))$. Since each $f_{1_k}(t \cdot (x_1 \leftarrow a))$ contains a set of non-dominated vectors, there must exist at least one combination of cost vectors $(\boldsymbol{v_1}, \ldots, \boldsymbol{v_{m_1}})$ where each $\boldsymbol{v_k} \in f_{1_k}(t \cdot (x_1 \leftarrow a))$, such that $C[1] = \sum_{k=1}^{m_1} \boldsymbol{v_k}$. Let S_a be the set of such combinations for domain value a (line 21). Tuple t_j is extended to variable x_1 with a domain value b for which exists at least one combination (line 22-23). One arbitrary combination $(\boldsymbol{v_1}, \ldots, \boldsymbol{v_{m_1}}) \in S_b$ is selected in line 24. The contribution to the solution of each $f_{1_k} \in \mathcal{B}_1$ is $\boldsymbol{v_k}$. Therefore, it is possible to update the array C of each departure bucket (line 25). The same procedure is repeated for each variable. At the end of the process, t_j is an efficient solution with cost vector $\boldsymbol{v_j}$.

Example 4. Consider the problem instance of example 3 where the weight and volume are not bounded. The weight and volume cost functions are, respectively:

$$w_i(x_i) = \begin{cases} 5 - i & x_i = 1 \\ 0 & x_i = 0 \end{cases} \qquad v_i(x_i) = \begin{cases} i & x_i = 1 \\ 0 & x_i = 0 \end{cases}$$

The objective functions to be minimized are,

$$\begin{aligned} &\mathcal{F}_1 = h_1 + h_2 + h_3 + \textstyle\sum_{x_i \in \mathcal{X}} f_i + f_{23} \\ &\mathcal{F}_2 = \textstyle\sum_{x_i \in \mathcal{X}} w_i \\ &\mathcal{F}_3 = \textstyle\sum_{x_i \in \mathcal{X}} p_i \end{aligned}$$

The trace of the algorithm under lexicographical ordering is:

- Input: the algorithm receives the seven trivially extended multi cost functions

$$h_1(x_1,x_3) = \begin{cases} (0,0,0) & x_1 \veebar x_3 \\ (\infty,0,0) & otherwise \end{cases} \qquad f_i(x_i) = \begin{cases} (0,0,0) & x_i = 1 \\ (i,0,0) & x_i = 0 \end{cases}$$

$$h_2(x_3,x_4) = \begin{cases} (0,0,0) & x_3 \vee x_4 \\ (\infty,0,0) & otherwise \end{cases} \qquad f_{23}(x_2,x_3) = \begin{cases} (0,0,0) & x_2 = 1 \wedge x_3 = 1 \\ (3,0,0) & \text{otherwise} \end{cases}$$

$$h_3(x_2,x_4) = \begin{cases} (0,0,0) & \overline{x_2 \wedge x_4} \\ (\infty,0,0) & otherwise \end{cases} \qquad w_i(x_i) = \begin{cases} (0,5-i,0) & x_i = 1 \\ (0,0,0) & x_i = 0 \end{cases}$$

$$v_i(x_i) = \begin{cases} (0,0,i) & x_i = 1 \\ (0,0,0) & x_i = 0 \end{cases}$$

- Elimination of x_4: $\mathcal{B}_4 = \{h_2, h_3, f_4, w_4, v_4\}$. Their sum is $b_4(x_2,x_3,x_4)$,

  $$b_4(001) = \{(0,1,4)\} \;\; b_4(010) = \{(4,0,0)\} \;\; b_4(011) = \{(0,1,4)\} \;\; b_4(110) = \{(4,0,0)\}$$

  Note that $b_4(000) = b_4(100) = b_4(101) = b_4(111) = \{\}$ because the sum of the multi cost functions in $\mathcal{B}_4$ for those tuples evaluates to $(\infty,0,0)$,$(\infty,0,0)$, $(0,\infty,4)$,$(\infty,1,4)$, respectively, and none of those cost vectors are consistent. In the sequel, we only indicate consistent evaluations.
  Projecting x_4 out of b_4 produces $g_4(x_3,x_4)$,

  $$g_4(00) = \{(0,1,4)\} \;\; g_4(01) = \{(4,0,0),(0,1,4)\} \;\; g_4(11) = \{(4,0,0)\}$$

- Elimination of x_3: $\mathcal{B}_3 = \{g_4, h_1, f_3, f_{23}, w_3, v_3\}$. Their sum is $b_3(x_1,x_2,x_3)$,

  $$b_3(001) = \{(7,2,3),(3,3,7)\} \;\; b_3(011) = \{(4,2,3)\} \;\; b_3(100) = \{(6,1,4)\}$$

  Projecting x_3 out of b_3 produces $g_3(x_1,x_2)$,

  $$g_3(00) = \{(7,2,3),(3,3,7)\} \;\; g_3(01) = \{(4,2,3)\} \;\; g_3(10) = \{(6,1,4)\}$$

- Elimination of x_2: $\mathcal{B}_2 = \{g_3, f_2, w_2, v_2\}$. Their sum is $b_2(x_1,x_2)$,

  $$b_2(00) = \{(9,2,3),(5,3,7)\} \;\; b_2(01) = \{(4,5,5)\} \;\; b_2(10) = \{(8,1,4)\}$$

  Projecting x_2 out of b_2 produces $g_2(x_1)$,

  $$g_2(0) = \{(9,2,3),(5,3,7),(4,5,5)\} \;\; g_2(1) = \{(8,1,4)\}$$

- Elimination of x_1: $\mathcal{B}_1 = \{g_2, f_1, w_1, v_1\}$. Their sum is $b_1(x_1)$,

  $$b_1(0) = \{(10,2,3),(6,3,7),(5,5,5)\} \;\; b_1(1) = \{(8,5,5)\}$$

  Projecting x_1 out of b_1 produces $g_1 = \{(10,2,3),(6,3,7),(5,5,5)\}$
  Note that $(8,5,5)$ is not a valid cost vector as it is dominated by $(5,5,5)$.

Therefore, the problem has three pareto optimal solutions. We show how to retrieve the one with costs $(10, 2, 3)$:

- Initially, $t = \lambda$, and $C[1] = (10, 2, 3)$.
- Variable x_1 assignment: There are two values for x_1,

$$\begin{array}{ll} t = (x_1 \leftarrow 0), & S_0 = \{((9,2,3),(1,0,0),(0,0,0),(0,0,0))\} \\ t = (x_1 \leftarrow 1), & S_1 = \{\} \end{array}$$

  Only value 0 satisfies the sum of multi cost functions in $\mathcal{B}_1$ because S_0 is not empty. Therefore, t is updated to $(x_1 \leftarrow 0)$ and the cost contribution of the departure bucket of every non original multi cost function in $\mathcal{B}_1$ is updated with its corresponding $\boldsymbol{v_j} \in (\boldsymbol{v_1}, \ldots, \boldsymbol{v_4})$. In this case, there is only one non original function, g_2. Therefore, $C[db(g_2)] = C[2] = (9, 2, 3)$.
- Variable x_2 assignment: There are two values for x_2,

$$\begin{array}{ll} t = (x_1 \leftarrow 0, x_2 \leftarrow 0), & S_0 = \{((7,2,3),(2,0,0),(0,0,0),(0,0,0))\} \\ t = (x_1 \leftarrow 0, x_2 \leftarrow 1), & S_1 = \{\} \end{array}$$

  Only value 0 satisfies the sum of multi cost functions in $\mathcal{B}_2$ because S_0 is not empty. Therefore, t is updated to $(x_1 \leftarrow 0, x_2 \leftarrow 0)$ and $C[db(g_3)] = C[3] = (7, 2, 3)$.
- Variable x_3 assignment: There are two values for x_3,

$$\begin{array}{ll} t = (x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 0), & S_0 = \{\} \\ t = (x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1), & S_1 = \{((4,0,0),(0,0,0),(0,0,0),(3,0,0),(0,2,0),(0,0,3))\} \end{array}$$

  Only value 1 satisfies the sum of multi cost functions in $\mathcal{B}_3$ as S_1 is not empty. Therefore, t is updated to $(x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1)$ and $C[db(g_4)] = C[4] = (4, 0, 0)$.
- Variable x_4 assignment: There are two values for x_4,

$$\begin{array}{ll} t = (x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1, x_4 \leftarrow 0), & S_0 = \{((0,0,0),(0,0,0),(4,0,0),(0,0,0),(0,0,0))\} \\ t = (x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1, x_4 \leftarrow 1), & S_1 = \{\} \end{array}$$

  Only value 0 satisfies the sum of multi cost functions in $\mathcal{B}_4$ because S_0 is not empty. Therefore, t is updated to $(x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1, x_4 \leftarrow 0)$. As there is no original function, the cost contribution vector C is not updated. As a result, the pareto optimal solution with objective vector $(10, 2, 3)$ is $(x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 1, x_4 \leftarrow 0)$.

Theorem 3. *MO-BE is space $O(n \times \prod_{i=1}^{p-1} K_i \times d^{w^*})$ and time $O(e \times \prod_{i=1}^{p-1} K_i^2 \times d^{w^*+1})$, where n is the number of variables, e is the number of cost functions, w^* is the graph induced width, K_i is the bound of each objective function, p is the number of objective functions, and d is the largest domain size.*

Proof. Let f be an arbitrary multi-cost function of arity r. Observe that its space complexity is $O(\prod_{i=1}^{p-1} K_i \times d^r)$ because: there are $O(d^r)$ different instantiations of the problem variables and, for each instantiation, there may be up to $O(\prod_{i=1}^{p-1} K_i)$ undominated instantiations. Since the largest arity among the functions that MO-BE needs to store is bounded by $O(w^*)$ and there are n such functions, the space and time complexities clearly hold.

Observe that K_p does not appear in the complexity of MO-BE. Since the order of the different objective functions is arbitrary, a straightforward optimization consists on leaving the largest K_i for the last position.

Property 1 *In a problem with a single objective function (i.e., p=1), the algorithm MO-BE is equivalent to BE.*

5 Application of MO-BE to WCSP with Global Constraints

It has been recently observed that there are some constraints (typically called global) that appear very frequently in constraint problems. The arity of these constraints is usually large and not to take into account their semantics during the solving process yields inefficient algorithms. A well known example is the enforcement of arc-consistency in the *all-diff* constraint: if arc-consistency is enforced with a generic algorithm the complexity is exponential on its arity, but there is a specialized algorithm that achieves the same result in quadratic time [**?**].

A similar situation has been detected when dealing with some constraints in the context of BE. Large arity constraints produces large cliques in the constraint graph which, in turn, imply a large induced width. Consequently, the space and time complexity of BE becomes prohibitive. In our previous work [**?**,**?**], we solved this problem for one important global constraint called knapsack. The idea was to make BE deal explicitly with knapsack constraints, exploiting their semantics. The new algorithm introduced so-called *state variables* to record useful information during the elimination of the variables. The benefit of such approach was that the constraint graph does not have to take into account the knapsack constraints. Therefore, they do not contribute by means of their (presumably large) scope, but only by their number. As a result, exponentially expensive problems with classical BE may become polynomial with the new algorithm.

In this Section we show that we can obtain exactly the same results with MO-BE. The idea is to reformulate global constraints as objective functions. We illustrate how to do it with *knapsack* constraints, *global cardinality* constraints and *all-diff* constraint.

Knapsack constraints (a.k.a. *capacity constraints*) arise in problems where some variables represent the potential uses of shared resources [**?**,**?**]. A *knapsack constraint* $C_i = (r_i, K_i)$ with scope $var(C_i) \subseteq \mathcal{X}$ bounds the use of a resource i. It is defined by a set of resource consumption functions $r_i = \{r_{ij} | x_j \in var(C_i)\}$ and the resource availability K_i. Let a be a domain value of $x_j \in var(C_i)$, then $r_{ij}(a)$ is the number of units of the resource i that a consumes when assigned to x_j. An assignment t of all variables in $var(C_i)$ satisfies the constraint if $\sum_{(x_j \leftarrow a) \in t} r_{ij}(a) < K_i$.

Consider a WCSP $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2, K)$ where the set of cost functions is formed by general cost functions $\mathcal{F}_1$ and knapsack constraints $\mathcal{F}_2 =$

$\{C_1, \ldots, C_e\}$. We can reformulate the problem into an equivalent MO-WCSP $\mathcal{P}' = (\mathcal{X}, \mathcal{D}, < \mathcal{F}_1, K > \cup < r_i, K_i >_{i=1}^{e})$. There is one objective function for each knapsack constraint C_i indicating that the consumption of resource i must be minimized and it is not acceptable to take more than K_i units. Besides, the set of general constraints in $\mathcal{F}_1$ define an additional objective function F_{e+1} subject to upper bound K. Once the efficient frontier Z of $\mathcal{P}'$ is computed, all vector components associated to knapsack constraints must be removed from every $\boldsymbol{v} \in Z$ leaving in Z a set of scalar values. Its minimum is the problem optimal solution.

The constraint graph of $\mathcal{P}$ will include the knapsack constraints. Thus, solving $\mathcal{P}$ with BE will be exponential in the induced width w^* where knapsack constraints are included in the constraint graph. On the other hand, the constraint graph of $\mathcal{P}'$ will not be affected by the knapsack constraints because their corresponding objective functions have unary cost functions only, which do not add arcs to the graph. Let w'^* denote the corresponding induced width. Solving $\mathcal{P}'$ with MO-BE will be time $O(\prod_{i=1}^{e} {K_i}^2 \times d^{w^*+1})$ and space $O(\prod_{i=1}^{e} K_i \times d^{w^*+1})$, where e is the number of capacity constraints, and d the largest domain size.

A *global cardinality constraint (gcc)* bounds the number of times values can be assigned to variables. This type of constraint commonly occurs in rostering, timetabling, sequencing, and scheduling applications [**?**].

For simplicity, we consider only one *gcc* over the whole set of variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ restricting values $\{a_1, \ldots, a_p\}$. Let lb_j and ub_j denote the bounds on the number of occurrences of value a_j. As already suggested in [**?**], the *gcc* can be modeled with $2p$-knapsack constrains. The first p knapsack constraints are $C_j = \{r_j, ub_j\}$ where $\forall k \leq n,\ r_{jk}(a)$ equals 1 when $a = a_j$, and 0 otherwise. They enforce the upper bounds, that is, the number of assignments of value a_j. The last p knapsack constraints are $C'_j = (r'_j, n - lb_j)$, where $\forall k \leq n,\ r'_{jk}(a)$ equals 0 when $a = a_j$, and 1 otherwise. They enforce the lower bounds, by reformulating them as upper bounds, that is, the number of assignments of values different from a_j.

Under this formulation, we can solve the problem using the same approach as in the previous subsection. However, MO-BE becomes more efficient than its worst-case complexity because in this particular case each knapsack constraint C_j is related to another constraint C'_j by the fact that $r_{jk}(a) = 1 - r'_{jk}(a)$. Then, function g_i, computed for each bucket i, satisfies that: if $(v_1, \ldots, v_p, v'_1, \ldots, v'_p, \ldots) \in g_i(t)$, then $\forall_{j=1..p} v'_j = n - i - v_j$. As a consequence, the size of each multi-cost function entry is $g_i(t) = O(\prod_{j=1}^{p} \min\{ub_j, n - lb_j\})$. The resulting complexity of MO-BE when solving a WCSP with one gcc is time $O(\prod_{j=1}^{p} \min\{ub_j, n - lb_j\}^2 \times d^{w^*+1})$ and space $O(\prod_{j=1}^{p} \min\{ub_j, n - lb_j\} \times d^{w^*})$.

A clear consequence of the previous complexity indicates that MO-BE is a suitable algorithm for WCSPs with one gcc, when the *gcc* restrict the number of occurrences of a small number of domain values, even if the restriction applies to a large number of variables.

An *all-different constraint* (all-diff) [**?**] is a well-known specialization of a *gcc* where n variables must take different values among n candidates. In this case, $lb_j = 0$ and $ub_j = 1$ for every domain value a_j. For simplicity, we consider a WCSP with only one all-diff constraint among all the variables. Following the previous ideas of the previous two Subsections, it can be reformulated as a MO-WCSP.

As the number of bounded domain values is n, the algorithm MO-BE is time $O(2^{2n} \times d^{w^*+1})$, which makes it worst than search. This result is hardly a surprise since it is well known that the traveling salesman problem is not suitable for dynamic programming techniques due to its implicit all-diff constraint that cannot be processed efficiently [**?**].

The approach presented along this Section has a potential source of inefficiency: an objective function still contributes to the algorithm space and time complexity even when its scope is totally processed. This problem can be overcome using the same idea describe in [**?**]. Namely, objective functions can be *deactivated* as soon as their scope is completely processed

6 Mini-Buckets with multi-objective WCSP

The concept of lower bound in mono-objective optimization can be extended to multi-objective optimization [**?**]. L is a *lower bound set* of R if,

1. $\forall \boldsymbol{v} \in R,\ \exists \boldsymbol{v'} \in L$ such that either $v = v'$ or $v' \leq v$.
2. $\forall \boldsymbol{v} \in R,\ \forall \boldsymbol{v'} \in L,\ \boldsymbol{v} \not< \boldsymbol{v'}$.

The idea of mini-buckets (MBE) can be adapted to the multiobjective situation in order to generate lower bound sets. The only difference between MBE and the new algorithm MO-MBE is that cost functions are replaced by multi cost functions. In MO-MBE large arity buckets are also split into subsets with a bounded arity. Each subset is processed separately, with bounded time and space complexity.

Theorem 4. *Executing MO-MBE with a MO-WCSP instance produces a lower bound set L of its efficient frontier Z.*

Proof. Consider bucket $\mathcal{B}_i$. MO-MBE creates a partition $Q' = \{Q_1, \ldots, Q_m\}$ of the bucket, where the join scope of each subset Q_l contains a bounded number of variables. MO-MBE processes each mini-bucket separately, thus computing a set of multi-cost functions g_{i_l},

$$g_{i_l} = (\sum_{f \in Q_{i_l}} f) \downarrow x_i$$

MO-BE, on the other hand, would compute the multi cost function g_i,

$$g_i = (\sum_{f \in \mathcal{B}_i} f) \downarrow x_i$$

First we show that $\sum_{j=1\ldots m} g_{i_j}$ is a lower bound set of g_i. Let $\boldsymbol{v} \in g_i(t)$ and $\boldsymbol{v'} \in \sum_{j=1\ldots m} g_{i_j}(t)$:

1. By definition of the projection of multi cost functions, there exists a domain value $a \in D_i$ such that $\boldsymbol{v} \in \sum_{f \in \mathcal{B}_i} f(t \cdot (x_i \leftarrow a))$. Therefore, $\boldsymbol{v} \in \sum_{l=1 \ldots m} \sum_{f \in Q_{i_l}} f(t \cdot (x_i \leftarrow a))$. As in each mini-bucket the value assigned to x_i should be different, either $\boldsymbol{v}$ or a vector $\boldsymbol{v}'$ that dominates $\boldsymbol{v}$ is an element of $\sum_{l=1 \ldots m} g_{i_l}(t)$. As a result, the first condition for a lower bound set follows.
2. Let $\boldsymbol{v} \in g_i(t)$. For the first property, either $\boldsymbol{v}$ or a vector $\boldsymbol{v'}$ that dominates $\boldsymbol{v}$ is an element of $\sum_{l=1 \ldots m} g_{i_l}(t)$. By definition, the sum of multi cost functions contains non-dominated vectors. As a result, if $\boldsymbol{v} \in \sum_{l=1 \ldots m} g_{i_l}(t)$, there is no $\boldsymbol{v'} \in \sum_{l=1 \ldots m} g_{i_l}(t)$ such that $\boldsymbol{v}$ *dominates* $\boldsymbol{v'}$. If $\boldsymbol{v} \notin \sum_{l=1 \ldots m} g_{i_l}(t)$, there is a $\boldsymbol{v'} \in \sum_{l=1 \ldots m} g_{i_l}(t)$ such that $\boldsymbol{v'}$ *dominates* $\boldsymbol{v}$. Therefore, there is no vector in $\sum_{l=1 \ldots m} g_{i_l}(t)$ dominated by any other vector in $g_i(t)$ and the second condition for a lower bound set also holds.

It is easy to see that the lower bound definition satisfies transitivity. Since MO-MBE processes buckets where all functions are either original or recursively processed by MO-MBE (which are lower bounds themselves), all functions computed by MO-MBE in a bucket are lower bound sets of the function that MO-BE would compute at that bucket.

Theorem 5. *MO-MBE, with accuracy parater k is space $O(e \times \prod_{i=1}^{p-1} K_i \times d^{k-1})$ and time $O(e \times \prod_{i=1}^{p-1} K_i^2 \times d^k)$, where e is the number of cost functions, K_i is the bound of each objective function, p is the number of objective functions, and d is the largest domain size.*

7 Computational Experiments

7.1 MO-BE for biobjective problems

In our first experiment we analize the suitability of MO-BE in multi-objective problems. To that end, we test MO-BE on two domains: $3-$MaxSat Max-Ones problems, and Vertex Cover problems with two objectives. We compare MO-BE with iterative search for the first domain, and MO-BE, iterative search and iterative integer programming for the second one.

A general schema for solving multi-objective problems is *iterative search*. The idea is to iteratively minimize one optimization criteria while bounding the second one. The procedure starts minimizing the first criteria with the second criteria bounded to its lowest possible value. The optimal value found in a given iteration is used as the upper bound for the next one. Moreover, the bound of second criteria is increased in one unit from iteration to iteration. The optimal value found in an iteration and the bound of the second criteria are a non-dominated point. The procedure ends when the highest value of the second criteria bound is reached. The result of iterative search is a set of non-dominated points, that is, the efficient frontier. Note that as the number of iterations are bounded by the maximum possible value of the criteria to be optimized, we select as the optimization criteria the one with minimum maximum value.

For the MaxSat Max-Ones domain, we run the approaches on instances from the dimacs benchmark []. Initially, such instances are $3-$MaxSat problems so that the optimization criteria is to minimize the number of violated clauses (i.e., the Max-Sat problem). We augment those problems with a second optimization criteria of minimizing the number of boolean variables taking value 0 (i.e., the Max-Ones problem).

We compare MO-BE with Iterative Ilog Solver (i.e., iterative search using Ilog Solver [] as search engine). Figure 4 shows the results for dimacs augmented problems with lower induced width. The last two columns reports the CPU time required for MO-BE and Iterative Ilog Solver with a time limit of 1800 seconds. It can be observed that for almost all instances, Iterative Ilog Solver is not able to find the efficient frontier within the time limit while MO-BE produces all non-dominated points very efficiently. Moreover, it can be seen that for those instances for which both approaches end, MO-BE outperforms Iterative Ilog Solver. Figure 5 depicts the non-dominated points that MO-BE and Iterative Ilog Solver found for $pret60\_25$ instance within the time limit.

The second domain where we test MO-BE is the Biobjective Minimum Vertex Cover. Given a graph G=(V,E) where each $v_i \in V$ has a weight w_i and is adjacent to other $v_j \in V$ (i.e., $e_{ij} \in E$), the *minimum vertex cover problem* is to find a subset S of vertices in G such that $\bigcup_{v_i \in S, e_{ij} \in E} e_{ij} = E$ minimizing $\sum_{v_i \in S} w_i$. In the biojective case, each $v_i \in V$ has two weights w_{i_1} and w_{i_2}, and therefore, there are two objective functions to be minimized. It is easy to see that Biobjective Minimum Vertex Cover can be expressed as a WCSP by representing vertexs as variables, edges as hard constraints between the two adjacent variables, and vertexs costs as unary cost functions which returns 0 if the vertex is not taken and its weight otherwise.

Biobjective Minimum Vertex Cover instances are generated with parameters (N, E, C_1, C_2) where N is the number of vertexs, E is the number of edges, C_1 is the maximum weight for the first vertex cost, and C_2 is the maximum weight for the second vertex cost. Instances are generated by selecting E pairs of vertexs at random and joining them. For each vertex, two costs are generated at random in the intervals $[0 \dots C_1]$ and $[0 \dots C_2]$, respectively.

We compare MO-BE, Iterative Ilog Solver and Iterative Integer Programming. The idea of Iterative Integer Programming is the same as for iterative search, the only difference being that Integer Programming is used instead of Constraint Programming. We use CPLEX [] as engine for Iterative Integer Programming. Figure 6 shows the results for Biobjective Vertex Cover problems. The last three columns reports the CPU time required for MO-BE, Iterative Ilog Solver and Iterative CPLEX with a time limit of 1800 seconds. As for the first domain, Iterative Ilog Solver exceeds the time limit for almost all parameter configuration. Iterative CPLEX finds the efficient frontier for all tests. However, MO-BE is still the best approach.

Instance	nb. vars	nb. clauses	w^*	nb. nondom sols	Time (sec.) MO-BE	It. Ilog Solver
dubois20	60	160	3	1	0	-
dubois21	63	168	3	1	0	-
dubois22	66	176	3	1	0	-
dubois23	69	184	3	1	0	-
dubois24	72	192	3	1	0	-
dubois25	75	200	3	1	0	-
dubois26	78	208	3	1	0	-
dubois27	81	216	3	1	0	-
dubois28	84	224	3	1	0	-
dubois29	87	232	3	1	0	-
dubois30	90	240	3	1	0	-
dubois50	150	400	3	1	0	-
dubois100	300	800	3	1	0.04	-
pret60_25	60	160	4	20	0	-
pret60_40	60	160	4	14	0	-
pret60_60	60	160	4	6	0	-
pret60_75	60	160	4	1	0	-
pret150_25	150	400	4	50	0	-
pret150_40	150	400	4	35	0	-
pret150_60	150	400	4	15	0	-
pret150_75	150	400	4	1	0	-
aim-50-1_6-no-1	50	76	15	8	3.17	-
aim-50-1_6-no-2	50	80	20	10	54.84	322
aim-50-1_6-no-3	50	80	19	10	15.59	-
aim-50-1_6-no-4	50	80	20	10	105.03	872
aim-50-1_6-yes1-1	50	80	18	10	22.97	654
aim-50-1_6-yes1-2	50	79	16	8	7.77	-
aim-50-1_6-yes1-3	50	80	19	10	64.64	362
aim-50-1_6-yes1-4	50	80	19	8	91.27	344
aim-50-2_0-no-4	50	98	23	10	1326.51	112
aim-50-2_0-yes1-3	50	100	23	14	1252.73	-
aim-50-2_0-yes1-4	50	100	21	14	297.26	954

Fig. 4. Experimental results on 3-MaxSat MaxOnes problems. Each problem is solved with MO-BE and Iterative Ilog Solver. Time limit 30 mins.

7.2 MO-MBE for biobjective problems

In our second experiment we evaluate the trade-off between accuracy and efficiency of MO-MBE. For that goal, we generate random 2-SAT instances with Allen van Gelder *mkcnf*'s generator[1]. We consider two simultaneous optimization criteria: The first one is to minimize the number of violated clauses (i.e, the Max-SAT problem). The second one is to minimize the number of boolean variables taking value 0 (i.e, the Max-Ones problem)

[1] ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/UCSC/instances

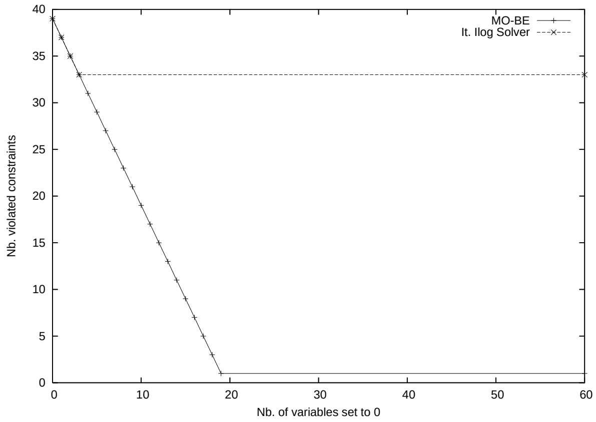

Fig. 5. Non-dominated points for *pret*60_25 instance found within the time limit with MO-BE and It. Ilog Solver.

N (nb. vars)	E (nb. edges)	w^*	nb. nondom sols	Time (sec.) MO-BE	It. Ilog Solver	It. CPLEX
60	95	8.4	8.35	0.0665	715.7	1.33
70	95	7.7	8.5	0.025	-	1.7
80	95	6.55	7.85	0.0565	-	2
90	95	5.05	9.65	0.031	-	2.1

Fig. 6. Experimental results on Biobjective Minimum Vertex Cover problems. Parameters C_1 and C_2 are both set to 5. Mean values on 20 instances for each parameter configuration. Time limit 30 mins.

We generated sets of instances with binary clauses, $n = \{10, 20, 25\}$, and a total number of different clauses corresponding to the $\{25\%, 50\%, 75\%, 100\%\}$ of the maximum number of combinations. For each parameter configuration samples of 50 instances were generated. The small size of these problems allowed solving them exactly with MO-BE.

Observe that the efficient frontier defines a region in the 2 dimensional space. One usual way to evaluate the accuracy of lower bound sets is to compute the percentage of area not covered by the lower bound set with respect the efficient set. Figure 8 shows average results for the area not covered by the lower bound set of MO-MBE with respect the efficient set (computed with MO-BE) for different values of k. It also reports the average CPU time in seconds for the different accuracies of MO-MBE. The first thing to be observed is that low values of k provide fairly good approximations (15-45% uncovered area) with very little computational efford. Increasing the value of k to higher values allows MO-MBE to compute much more accurate lower bound sets.

Observe that when increasing the number of clauses, the accuracy of MO-MBE with the same value of k also increases (i.e., the proportion of uncovered area decreases). The reason for that behavior is that a low number of clauses means that each objective is fairly easy to satisfy and all efficient solutions have low values for every objective. Since costs are natural numbers, the region covered by the efficient frontier has a coarse granularity. Thus, even a close sub-optimal solution will be of bad quality if we measure it as the proportion of uncovered area.

7.3 MO-MBE for mono-objective problems

In our second experiment we analyze the applicability of MO-MBE in mono-objective optimization problems with knapsack constraints. To that aim, we run MO-MBE on instances from the Spot5 benchmark [?]. They are taken from the daily scheduling of an earth observation satellite. The problem is to schedule a subset of photographs from a set of candidates, which will be effectively taken by the satellite. The resulting subset of photographs must satisfy a large number of hard constraints and at the same time maximize the importance of selected photographs[2]. Some instances include a capacity constraint expressing the maximum recording capacity on board of the satellite which cannot be surpassed (each photograph has an associated storage requirement). While all the instances without capacity constraint have been solved, all the instances with capacity constraint but two remain unsolved. We experiment with unsolved instances with capacity constraint.

All the instances are encoded as WCSPs: Potential photographs are the variables. The different ways to take a photograph are the different domain values (an additional value is added to each variable meaning that the photograph is not taken). Hard constraints are transformed into cost functions that return cost ∞ to infeasible assignments and 0 otherwise. The importance of each photograph is encoded with a unary cost function which returns 0 if the photograph is taken and its importance if the photograph is not taken. The scope of the capacity constraint is the whole set of variables.

One possible way to compute lower bounds consists on removing the capacity constraint from the instances (the optimum of this relaxation will obviously be less than or equal to the optimum in the original problem) and then execute classical MBE. An alternative is to reformulate the problem as bi-objective. The sum of all cost functions constitute the first objective function F_1. The capacity constraint is added as a second objective function F_2. We can execute MO-MBE.

We compare these two approaches. Figure 9 reports the lower bounds obtained for different values of the accuracy parameter k as well as the CPU time required for each execution. It can be observed that for all instances MO-MBE produces much higher lower bounds than MBE. While this is clearly true if we compare executions with the same value of k, such comparison is not totally fair because MO-MBE has a higher complexity due to the computation of multi-cost functions. However, if we look at executions with a similar CPU time we still observe a clear dominance of MO-MBE.

[2] Without loss of generality, in the following we will assume a minimization problem where the objective function is to minimize the importance of discarded photographs

8 Conclusion and Future Work

Problems involving the optimization of more than one objective are ubiquitous in real world domains and little considered in the constraint community. In this paper we have extended the *weighted constraint satisfaction problem* (WCSP) to *multiobjective* optimization (MO-WCSP). Then, we have generalized bucket elimination (BE) and mini-bucket elimination (MBE) to deal with MO-WCSP. We have shown that MO-BE can be used to solve true multiobjective problems with bounded induced width, as well as mono-objective problems with knapsack-like constrains when the induced width disregarding the knapsack constraints is bounded. In the latter case, MO-BE may produce exponential savings over the usual application of BE over the original mono-objective problem where the knapsack constraints are taken into consideration in the induced graph. MO-MBE can be used to compute lower bound sets of multiobjective problems. The accuracy parameter k of the mini-buckets technique allows two potential uses of MO-MBE. With high values of k, it can be used to obtain good quality lower bounds of problems that cannot be solved exactly. It can be used in combination with metaheuristic algorithms in order to bound the actual efficient frontier. With low values of k, it can be used inside a branch and bound solver to increase its pruning capability.

In our current research we are considering the extension of MO-BE to the more general framework of *Semiring-based* CSPs which allows a much more general specification of multiobjective optimization. We are also interested in detecting other global constrains that can be processed with multiobjective algorithms. Finally, we want to evaluate the practical potential of MO-BE for problems with bounded induced width and branch and bound with MO-MBE for problems with unbounded induced width.

Instance	nb. vars	nb. clauses	w^*	k	nb. nondom sols	Time (sec.) MO-MBE	Time (sec.) It. Ilog Solver
aim-100-1_6-no-1	100	160	40.	5	6	0	-
				10	7	0.08	-
				15	10	4.3	-
				20	12	254.31	-
aim-100-1_6-no-2	100	160	37.	5	4	0	-
				10	7	0.09	-
				15	9	3.82	-
				20	13	232.89	-
aim-100-1_6-no-3	100	157	40.	5	7	0	-
				10	11	0.08	-
				15	11	4.11	-
				20	13	180.17	-
aim-100-1_6-no-4	100	160	38.	5	6	0	-
				10	8	0.09	-
				15	15	3.88	-
				20	11	203.34	-
aim-100-1_6-yes1-1	100	160	36.	5	7	0	-
				10	9	0.1	-
				15	15	5.6	-
				20	15	276.34	-
aim-100-1_6-yes1-2	100	160	33.	5	10	0	-
				10	8	0.09	-
				15	12	5.3	-
				20	14	199.24	-
aim-100-1_6-yes1-3	100	160	35.	5	9	0	-
				10	10	0.08	-
				15	13	5.41	-
				20	13	229.05	-
aim-100-1_6-yes1-4	100	160	37.	5	10	0	-
				10	11	0.09	-
				15	14	4.3	-
				20	17	181.87	-
aim-100-2_0-no-1	100	200	51.	5	5	0	-
				10	5	0.09	-
				15	9	4.11	-
				20	9	200.26	-
aim-100-2_0-no-2	100	199	53.	5	6	0	-
				10	8	0.09	-
				15	10	3.67	-
				20	8	193.93	-
aim-100-2_0-no-3	100	198	51.	5	4	0	-
				10	6	0.08	-
				15	5	3.72	-
				20	8	235.93	-
aim-100-2_0-no-4	100	200	48.	5	4	0	-
				10	5	0.08	-
				15	7	4.06	-
				20	6	197.7	-
aim-100-2_0-yes1-1	100	198	45.	5	9	0	-
				10	14	0.12	-
				15	16	3.91	-
				20	17	295.23	-
aim-100-2_0-yes1-1	100	198	45.	5	9	0	-
				10	14	0.1	-
				15	16	3.9	-
				20	17	215.47	-
aim-100-2_0-yes1-3	100	200	47.	5	7	0	-
				10	12	0.1	-
				15	14	4.32	-
				20	16	284.33	-

Instance	nb. vars	nb. clauses	w^*	k	nb. nondom sols	Time (sec.) MO-MBE	It. Ilog Solver
aim-100-2_0-yes1-4	100	199	44.	5	7	0	-
				10	12	0.11	-
				15	12	4.89	-
				20	16	308.9	-
aim-200-1_6-no-1	200	320	89.	5	12	0	-
				10	9	0.16	-
				15	7	6.58	-
				20	12	322.36	-
aim-200-1_6-no-2	200	317	84.	5	6	0	-
				10	10	0.18	-
				15	13	6.69	-
				20	10	361.58	-
aim-200-1_6-no-3	200	320	74.	5	11	0	-
				10	12	0.14	-
				15	12	5.96	-
				20	14	238.95	-
aim-200-1_6-no-4	200	320	91.	5	10	0	-
				10	9	0.17	-
				15	8	5.67	-
				20	12	290.13	-
aim-200-1_6-yes1-1	200	320	74.	5	16	0	-
				10	19	0.21	-
				15	17	10.89	-
				20	23	332.5	-
aim-200-1_6-yes1-2	200	320	66.	5	22	0	-
				10	21	0.21	-
				15	18	7.46	-
				20	20	435.72	-
aim-200-1_6-yes1-3	200	319	76.	5	20	0	-
				10	20	0.2	-
				15	25	10.07	-
				20	28	420.72	-
aim-200-1_6-yes1-4	200	320	67.	5	9	0	-
				10	12	0.16	-
				15	19	7.5	-
				20	21	472.75	-
aim-200-2_0-yes1-3	200	399	-	5	14	0.01	-
				10	19	0.26	-
				15	17	10.07	-
				20	21	364.33	-
jnh01	100	850	95.	5	5	0.01	-
				10	6	0.3	-
				15	9	12.2	-
				20	8	462.97	-
jnh02	100	850	94.	5	3	0.01	-
				10	6	0.28	-
				15	8	11.32	-
				20	13	573.96	-
jnh03	100	850	95.	5	3	0.01	-
				10	7	0.26	-
				15	11	9.62	-
				20	9	435.58	-
jnh04	100	850	95.	5	2	0.01	-
				10	5	0.25	-
				15	7	8.93	-
				20	7	396.27	-
jnh05	100	850	94.	5	5	0.01	-
				10	12	0.3	-
				15	12	11.2	-
				20	13	472.17	-

Instance	nb. vars	nb. clauses	w^*	k	nb. nondom sols	Time (sec.) MO-MBE	It. Ilog Solver
jnh06	100	850	95.	5	3	0.01	-
				10	5	0.27	-
				15	8	10.37	-
				20	11	359.11	-
jnh07	100	850	-	5	4	0.01	-
				10	6	0.27	-
				15	7	9.37	-
				20	10	454.6	-
jnh08	100	850	93.	5	6	0.01	-
				10	7	0.28	-
				15	6	10.22	-
				20	9	429.49	-
jnh09	100	850	94.	5	1	0.01	-
				10	6	0.24	-
				15	6	10.4	-
				20	11	458.99	-
jnh10	100	850	94.	5	7	0.01	-
				10	7	0.28	-
				15	12	12.09	-
				20	12	449.7	-
jnh11	100	850	94.	5	3	0.01	-
				10	6	0.24	-
				15	8	8.74	-
				20	10	342.07	-
jnh12	100	850	94.	5	4	0.01	-
				10	5	0.25	-
				15	10	10.19	-
				20	12	422.09	-
jnh13	100	850	94.	5	4	0.01	-
				10	9	0.27	-
				15	14	11.42	-
				20	14	467.7	-
jnh14	100	850	94.	5	4	0.01	-
				10	3	0.25	-
				15	5	7.84	-
				20	5	322.32	-
jnh16	100	850	93.	5	4	0.01	-
				10	7	0.29	-
				15	9	10.65	-
				20	13	462.88	-
jnh17	100	850	94.	5	5	0.01	-
				10	8	0.26	-
				15	7	10.58	-
				20	8	413.02	-
jnh18	100	850	94.	5	5	0.01	-
				10	8	0.27	-
				15	13	10.56	-
				20	13	516.68	-
jnh201	100	800	93.	5	6	0.01	-
				10	4	0.24	-
				15	8	11.07	-
				20	7	472.71	-
jnh202	100	800	-	5	2	0.01	-
				10	6	0.26	-
				15	5	8.98	-
				20	10	438.79	-
jnh203	100	800	94.	5	7	0.01	-
				10	8	0.26	-
				15	12	12.83	-
				20	11	607.16	-

Instance	nb. vars	nb. clauses	w^*	k	nb. nondom sols	Time (sec.) MO-MBE	It. Ilog Solver
jnh204	100	800	95.	5	4	0.01	-
				10	5	0.24	-
				15	7	9.46	-
				20	9	384.28	-
jnh205	100	800	93.	5	7	0.01	-
				10	9	0.27	-
				15	9	10.71	-
				20	11	392.35	-
jnh206	100	800	95.	5	3	0.01	-
				10	10	0.24	-
				15	11	10.63	-
				20	12	500.42	-
jnh208	100	800	94.	5	7	0.01	-
				10	9	0.29	-
				15	8	9.55	-
				20	12	489.35	-
jnh20	100	850	94.	5	3	0.01	-
				10	9	0.29	-
				15	13	12.58	-
				20	14	445.16	-
jnh210	100	800	94.	5	1	0.01	-
				10	6	0.24	-
				15	8	10.12	-
				20	10	368.27	-
jnh211	100	800	93.	5	8	0.01	-
				10	6	0.27	-
				15	11	11.58	-
				20	12	504.29	-
jnh212	100	800	97.	5	2	0.01	-
				10	5	0.27	-
				15	6	8.7	-
				20	6	347.89	-
jnh213	100	800	93.	5	5	0.01	-
				10	7	0.25	-
				15	15	10.87	-
				20	14	602.66	-
jnh215	100	800	94.	5	4	0.01	-
				10	6	0.24	-
				15	9	10.26	-
				20	7	359.52	-
jnh217	100	800	93.	5	5	0.01	-
				10	10	0.26	-
				15	12	9.91	-
				20	12	396.43	-
jnh218	100	800	93.	5	5	0.01	-
				10	8	0.28	-
				15	7	9.67	-
				20	10	430.02	-
jnh219	100	800	93.	5	6	0.01	-
				10	10	0.25	-
				15	9	9.69	-
				20	13	387.37	-
jnh301	100	900	95.	5	6	0.01	-
				10	3	0.29	-
				15	3	10.78	-
				20	6	443.31	-
jnh302	100	900	94.	5	5	0.01	-
				10	7	0.29	-
				15	6	10.64	-
				20	9	406.58	-

Instance	nb. vars	nb. clauses	w^*	k	nb. nondom sols	Time (sec.) MO-MBE	It. Ilog Solver
jnh303	100	900	94.	5	5	0.01	-
				10	7	0.29	-
				15	9	11.82	-
				20	12	542.44	-
jnh304	100	900	95.	5	3	0.01	-
				10	6	0.29	-
				15	9	10.75	-
				20	10	430.26	-
jnh305	100	900	94.	5	7	0.01	-
				10	10	0.29	-
				15	11	11.68	-
				20	14	561.71	-
jnh306	100	900	95.	5	2	0.01	-
				10	6	0.27	-
				15	8	10.54	-
				20	9	428.19	-
jnh307	100	900	95.	5	5	0.01	-
				10	8	0.32	-
				15	11	12.25	-
				20	13	504.39	-
jnh308	100	900	-	5	5	0.01	-
				10	4	0.27	-
				15	7	9.53	-
				20	9	445.67	-
jnh309	100	900	94.	5	2	0.01	-
				10	6	0.28	-
				15	10	10.9	-
				20	12	485.69	-
jnh310	100	900	94.	5	5	0.01	-
				10	7	0.27	-
				15	8	10.27	-
				20	14	556.62	-
ssa0432-003	435	1027	18.	5	123	0.09	-
				10	125	5.29	-
				15	133	265.44	-
				20	144	2126.44	-
ssa2670-130	1359	3321	23.	5	281	0.95	-
				10	342	54.33	-
				15	349	1510.31	-
ssa7552-038	1501	3575	25	5	254	15.65	
				10	357	128.32	
				15	379	4318.04	
				5	237	13.9	
ssa7552-158	1363	3034		10	10	327	968.73
				15	329	1058.11	
				5	239	4.03	
ssa7552-159	1363	3032		11	10	328	104.32
				15	329	124.01	
				5	228	18.5	
ssa7552-160	1391	3126		14	10	327	99.07
				15	335	5164.81	

Fig. 7. Experimental results on 3-MaxSat MaxOnes problems. Each problem is solved with MO-BE and Iterative Ilog Solver. Time limit 30 mins.

2-MaxSat-MaxOnes: mean values on 50 instances							
		$n = 10$		$n = 20$		$n = 25$	
nb. clauses	k	%U	time	%U	time	%U	time
25%	3	32.09	0.0006	42.76	0.0014	45.59	0.0028
	6	10.74	0.0008	22.75	0.0062	28.89	0.0086
	9	1.31	0.0006	12.36	0.033	18.09	0.061
	12			6.40	0.2136	9.19	0.461
	15			2.85	1.2234	5.47	3.4168
50%	3	22.47	0.0004	31.06	0.0056	25.12	0.006
	6	6.99	0.0002	16.93	0.011	13.79	0.0174
	9	0.95	0.0018	8.54	0.062	7.02	0.1322
	12			4.49	0.3904	3.50	1.0032
	15			2.12	2.4714	2.03	7.746
75%	3	18.47	0.0012	25.02	0.0056	26.12	0.0085
	6	4.43	0.0004	13.93	0.0146	16.37	0.02375
	9	0.78	0.0048	7.57	0.0954	10.42	0.1645
	12			3.48	0.6066	4.94	1.38
	15			1.93	3.8558	2.75	9.574
100%	3	15.40	0.001	21.21	0.0084	22.60	0.0138776
	6	5.43	0.0006	11.65	0.0212	14.21	0.0346939
	9	0.58	0.0098	6.06	0.133	8.84	0.218776
	12			3.04	0.8154	4.42	1.75633
	15			1.48	5.287	2.53	13.489

Fig. 8. Trade-off between accuracy and efficiency of MO-MBE(k) on 2-MaxSat-MaxOnes.

				MO-MBE(k)		MBE(k)	
Instance	nb. vars	nb. constr.	k	cost	time	cost	time
1506	940	14301	5	244433	0.31	123174	0.16
			10	249479	5.04	138216	1.36
			15	254513	124.92	167267	25.74
1401	488	10476	5	327152	0.25	125059	0.12
			10	333151	5.19	137060	0.97
			15	343145	154.66	165064	21.57
1403	665	12952	5	326249	0.3	121123	0.15
			10	339265	5.58	137131	1.22
			15	340267	156.51	169144	26.3
1405	855	17404	5	322426	0.38	117170	0.22
			10	334436	7.25	150171	1.69
			15	341452	153.2	171195	35.58
1407	1057	20730	5	321475	0.43	118172	0.28
			10	342519	6.67	147205	2.06
			15	345543	281.7	175250	41.23

Fig. 9. Experimental results on Spot5 instances. Each problem is solved with MO-MBE(k) and MBE(k) with different values of k.