J Heuristics (2006) 12: 307–328
DOI 10.1007/s10732-006-6726-y

Bucket elimination for multiobjective optimization problems

Emma Rollón · Javier Larrosa

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 multiobjective 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 Max-SAT-ONE and biobjective weighted minimum vertex cover problems.

Keywords Multiobjective optimization · Dynamic programming · Decomposition methods · Global constraints

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 (Dechter, 2003). 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 (Bistarelli et al., 1999). Thus, CSPs and soft CSPs are decision and optimization problems, respectively. 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 (Freuder and Wallace, 1992)) or dynamic programming (e.g., bucket elimination, BE (Dechter, 1999)). Approximate algorithms can compute

E. Rollón · J. Larrosa (✉)
Universitat Politecnica de Catalunya, Barcelona, Spain
e-mail: {erollon, larrosa}@lsi.upc.edu

upper bounds (e.g., metaheuristics (Bertrand and Trombettoni, 2003)) or lower bounds (e.g. relaxation methods (de Givry, Verfaillie, and Schiex, 1997). Algorithms that compute lower bounds (such as mini-bucket elimination, MBE (Dechter and Rish, 2003)) are a fundamental component of branch and bound because they can be executed at every search node in order to detect infeasible nodes (Meseguer et al., 2001).

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 (Ehrgott and Gandibleux, 2002)) 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 focuses on specific techniques for 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 introduce 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 the 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 with knapsack-like constraints. Moreover, we show that this approach provides the same complexity improvement as our previous work (Larrosa and Rollon, 2004*a*, 2004*b*) 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 biobjective and real scheduling problems.

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 $x_1 \underline{\vee} x_3$ ($\underline{\vee}$ means *exclusive or*). The problem is a CSP with four solutions: 0010, 0011, 0110 and 1001.

Weighted CSP (WCSP) (Bistarelli et al., 1997; Schiex et al., 1995) 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,

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

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

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

Soft constraints encoded as unary and binary functions,

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

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 $w^*(o)$ (Dechter, 2003).

2.1. Bucket elimination

Bucket elimination (BE) (Dechter, 1999; Bertele and Brioschi, 1972) 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 $O(d^{|var(f)|})$.

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

Fig. 1 Bucket Elimination. Given a WCSP $(\mathcal{X}, D, F, K)$, 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 algorithm returns NIL

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

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. *(Dechter, 1999) 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

Fig. 2 Second phase of the bucket elimination with a non-standard implementation

```
7.  $t := \lambda$;
8.  $C[1] := g_1$;
9.  for each $i = 1..n$ do
10.     let $\mathcal{B}_i = \{f_{i_1}, f_{i_2}, \dots, f_{i_{m_i}}\}$
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.     $(v_1, \dots, v_{m_i}) := (f_{i_1}(t \cdot (x_i \leftarrow b)), \dots, f_{i_{m_i}}(t \cdot (x_i \leftarrow b)))$;
14.     $\forall 1 \le k \le m_i$ if $db(f_{i_k}) \ne i$ then $C[db(f_{i_k})] := v_k$;
15. endfor
16. return$(g_1, t)$;
```

function was generated. Therefore, $db(f_{i_k}) = i$ if f_{i_k} is an original function, and $db(f_{i_k}) = j$ if $f_{i_k} = g_j$.

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 $C[1 \dots n]$. 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) (Dechter and Rish, 2003) 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 $Q' = \{Q_1, \dots, Q_m\}$ of the functions in $\mathcal{B}_i$. The approximation processes each subset separately, thus computing a set of functions $\{g_{il}\}_{l=1}^m$,

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

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

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

Since,

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

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. *(Dechter and Rish, 2003) 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. Multiobjective weighted constraint satisfaction problem

A *multiobjective* WCSP (MO-WCSP) is defined as $(\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_i, K_i \rangle_{i=1}^{p})$, 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 $\mathcal{EF}$ be their set of non-dominated costs, also called the *efficient frontier*. The task in a MO-WCSP is to compute $\mathcal{EF}$ (and, possibly, one or all efficient solutions for each of its elements). Observe that $|\mathcal{EF}| = O(\prod_{i=1}^{p-1} K_i)$. 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 $|\mathcal{EF}| = O(1)$. As in the WCSP case, each MO-WCSP instance has an associated constraint graph G computed in the same way. Note that the constraint graph G contains arcs coming from all the individual objective functions.

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, $f(t) = \{\mathbf{v}_1, \ldots, \mathbf{v}_r\}$, with $\mathbf{v}_i = (v_{i1}, \ldots, v_{ip})$, such that $\forall\ i, j,\ v_{ij} < K_j$ and $\forall\ i, j,\ \mathbf{v}_i \not< \mathbf{v}_j$. Observe that $|f(t)| = O(\prod_{i=1}^{p-1} K_i)$ and $|f| = O(\prod_{i=1}^{p-1} K_i \times d^{|var(f)|})$.

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,

$$h(t) = \{\boldsymbol{v} \mid \ t = t' \cdot t'', \boldsymbol{v} = \boldsymbol{v}' + \boldsymbol{v}'', \boldsymbol{v}' \in f(t'), \boldsymbol{v}'' \in g(t''),$$

$$\texttt{consistent}(\boldsymbol{v}), \texttt{nondominated}(\boldsymbol{v})\}$$

  $f + g$ 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,

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

  $h = f \downarrow x_i$ 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.

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

```
function MO-BE($\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_i, K_i \rangle_{i=1}^{p}$)
1.  $\mathcal{F} := \emptyset$
2.  for each $i = 1 \ldots p$, $f \in F_i$ do
3.    for each $t \in var(f)$ do
4.      if $f(t) \geq K_i$ then $h(t) := \emptyset$
5.      else $h(t) := \{(v_1, \ldots, v_p)\}$ where $v_i = f(t)$ and $v_j = 0, \forall j \neq i$
6.    endfor
7.    $\mathcal{F} := \mathcal{F} \cup \{h\}$
8.  endfor
9.  for each $i = n \ldots 1$ do
10.   $\mathcal{B}_i := \{h \in \mathcal{F} \mid x_i \in var(h)\}$;
11.   $g_i := (\sum_{h \in \mathcal{B}_i} h) \downarrow x_i$;
12.   if $\forall t, g_i(t) = \emptyset$ then return NIL;
13.   $\mathcal{F} := (\mathcal{F} \cup \{g_i\}) - \mathcal{B}_i$;
14. endfor
15. let $g_1 = \{\boldsymbol{v_1}, \boldsymbol{v_2}, \ldots, \boldsymbol{v_r}\}$
16. for each $j = 1 \ldots r$ do
17.   $t_j := \lambda$;
18.   $C[1] := \boldsymbol{v_j}$
19.   for each $i = 1 \ldots n$ do
20.     let $\mathcal{B}_i = \{h_{i_1}, \ldots, h_{i_{m_i}}\}$
21.     $\forall a \in D_i \ S_a = \{(\boldsymbol{v_1}, \ldots, \boldsymbol{v_{m_i}}) \mid \sum_{k=1}^{m_i} \boldsymbol{v_k} = C[i], \ \forall k, \ \boldsymbol{v_k} \in h_{i_k}(t \cdot (x_i \leftarrow a))\}$
22.     $b := \text{pop}(\{a \in D_i \mid S_a \neq \emptyset\})$
23.     $t_j := t_j \cdot (x_i \leftarrow b)$;
24.     $(\boldsymbol{v_1}, \ldots, \boldsymbol{v_{m_i}}) := \text{pop}(S_b)$
25.     $\forall 1 \leq k \leq m_i$ if $db(h_{i_k}) \neq i$ then $C[db(h_{i_k})] := \boldsymbol{v_k}$;
26.   endfor
27. endfor
28. 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 $(\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_i, K_i \rangle_{i=1}^{p})$. 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

instance $(\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_i, K_i \rangle_{i=1}^{p})$. 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 $h(t) = \{(v_1, v_2, \dots, v_p)\}$, 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 $\mathcal{EF}$. It works as BE, the only difference being that multi cost functions are used instead of standard cost functions. Let $\boldsymbol{v} = (v_1, \dots, v_p)$ 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 $x_i, x_{i+1}, \dots, x_n$ 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 $\mathcal{EF}$ of the problem.

Let g_1 contain r vector points $\{\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_r}\}$. 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 $x_1, \dots, x_n$ (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 $(\boldsymbol{v_1}, \dots, \boldsymbol{v_{m_1}})$ 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 $(\boldsymbol{v_1}, \dots, \boldsymbol{v_{m_1}}) \in S_b$ 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

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

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

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

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

- 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 $(\infty,0,0)$, $(\infty,0,0)$, $(\infty,1,4)$, $(\infty,1,4)$, 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 $g_4(x_2,x_3)$,

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

- 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,

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

  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,

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

  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,

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

  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,

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

  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. $\square$

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 (Regin, 1994).

A similar situation has been detected when dealing with some constraints in the context of BE. Large arity constraints produce 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 (Larrosa and Rollon, 2004*b*, 2004*a*), 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 (Trick, 2003; Sellmann, 2003). 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 $\mathcal{P}' = (\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_1, K \rangle \cup \langle r_i, K_i \rangle_{i=1}^{e})$. 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 $\mathcal{EF}$ of $\mathcal{P}'$ is computed, all vector components associated to knapsack constraints must be removed from every $\boldsymbol{v} \in \mathcal{EF}$ leaving in $\mathcal{EF}$ 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 (Quimper et al., 2004).

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 (van Hentenryck et al., 1992), the *gcc* can be modeled with $2p$-knapsack constraints. 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 for knapsack constraints. 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) (Regin, 1994) 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 ideas for the previous two global constraints, 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 worse 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 (Brassard and Bratley, 1995).

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 described in (Larrosa and Rollon, 2004a). 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 (Ehrgott and Gandibleux, 2001). L is a *lower bound set* of R if,

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

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 $\mathcal{EF}$.*

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} □

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

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

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

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

1. By definition of the projection of multi cost functions, there exists a domain value $a \in D_i$ such that $\mathbf{v} \in \sum_{f \in \mathcal{B}_i} f(t \cdot (x_i \leftarrow a))$. Therefore, $\mathbf{v} \in \sum_{l=1\ldots m} \sum_{f \in Q_{i_l}} f(t \cdot (x_i \leftarrow a))$. As in each mini-bucket the value assigned to x_i should be different, either $\mathbf{v}$ or a vector $\mathbf{v}'$ that dominates $\mathbf{v}$ 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 $\mathbf{v} \in g_i(t)$. For the first property, either $\mathbf{v}$ or a vector $\mathbf{v'}$ that dominates $\mathbf{v}$ 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 $\mathbf{v} \in \sum_{l=1\ldots m} g_{i_l}(t)$, there is no $\mathbf{v'} \in \sum_{l=1\ldots m} g_{i_l}(t)$

such that v *dominates* v'. If $v \notin \sum_{l=1\ldots m} g_{i_l}(t)$, there is a $v' \in \sum_{l=1\ldots m} g_{i_l}(t)$ such that v' *dominates* 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 parameter 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. Biobjective iterative deepening

One way to solve multi-objective problems with mono-objective technology is to extend iterative deepening (Korf, 1985). The idea is to bound the different objective functions except one and iteratively increase the bounds. For instance, if we have a mono-objective optimization algorithm and a biobjective problem, we can use iterative deepening on one of the objectives and, for each bound, optimize the other. This algorithm, that we call BID, is depicted in Figure 4. It computes the efficient frontier sequentially. In our experiments we used Solver 6.1 to solve the mono-objective problem of line 5 with default heuristics for variable selection.

If the multi-objective problem is too difficult for the previous approach, we can use iterative deepening on every objective function. If the available time is exhausted, we have a lower bound of the efficient frontier. This algorithm, that we call BID-lb, is depicted in Figure 5. It computes an increasingly large lower bound. As before, we used Solver 6.1 to solve the decision problem of line 10 with default heuristics for variable selection.

We use BID and BID-lb as algorithms of reference to evaluate the performance of MO-BE and MO-MBE, respectively.

```
function Biobjective_Iterative_Deepening($\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_i, K_i \rangle_{i=1}^2$) return set of pairs
1.  $\mathcal{EF} := \emptyset$
2.  $k_1 := 0$;  $k_2 := K_2$;
3.  while $k_1 < K_1$ and $k_2 \neq 0$ do
4.      $k_1 := k_1 + 1$;
5.      $y :=$ minimize $\mathcal{F}_2(X)$ subject to $\mathcal{F}_1(X) < k_1$ and $\mathcal{F}_2(X) < k_2$;
6.      if $y \neq NIL$ then
7.          $k_2 := y$;  $\mathcal{EF} :=$ non-dominated($\mathcal{EF} \bigcup \{(k_1, k_2)\}$);
8.      endif
9.  endwhile
10. return $\mathcal{EF}$;
endfunction
```

Fig. 4 Biobjective iterative deepening (BID). Given a MO-WCSP $(\mathcal{X}, D, \langle \mathcal{F}_i, K_i \rangle_{i=1}^2)$, the algorithm returns the efficient frontier $\mathcal{EF}$. The function *non-dominated ($\mathcal{EF}$)* eliminates the dominated pairs from its input parameter $\mathcal{EF}$. Note that if the task in line 5 returns NIL, k_2 is its solution and (k_1, k_2) is dominated by one point already included in $\mathcal{EF}$

```
function Biobjective_Iterative_Deepening_Lower_Bound($(\mathcal{X}, \mathcal{D}, \langle \mathcal{F}_i, K_i \rangle_{i=1}^2)$)
return set of pairs
1.  $\mathcal{EF} := \emptyset$; $\mathcal{LB} := \emptyset$
2.  $i := 0$;
3.  while $i < K_2 + K_1$ do
4.      $i := i + 1$;
5.      $k_1 := 0$;
6.      $k_2 := i$;
7.      while $k_2 > 0$ do
8.          $k_1 := k_1 + 1$;
9.          if $\forall \boldsymbol{v} \in \mathcal{EF}, \boldsymbol{v} \not< (k_1, k_2)$ then
10.             if solve $(\mathcal{X}, \mathcal{D}, \mathcal{F}_1 \bigcup \mathcal{F}_2)$ subject to $\mathcal{F}_1(X) < k_1$ and $\mathcal{F}_2(X) < k_2$ then
11.                 $\mathcal{EF} :=$ non-dominated($\mathcal{EF} \bigcup \{(k_1, k_2)\}$);
12.             else $\mathcal{LB} :=$non-dominated($\mathcal{LB} \bigcup \{(k_1, k_2)\}$);
13.         endif
14.         $k_2 := k_2 - 1$;
15.     endwhile
16. endwhile
17. return $\mathcal{EF} \bigcup \mathcal{LB}$;
endfunction
```

Fig. 5 Biobjective iterative search lower bound (BID-lb). Given a MO-WCSP $(\mathcal{X}, D, \langle \mathcal{F}_i, K_i \rangle_{i=1}^2)$, the algorithm returns a lower bound set. Note that if $\mathcal{LB} = \emptyset$, $\mathcal{EF}$ is the complete efficient frontier of the problem. Moreover, if $\mathcal{LB} \neq \emptyset$ and $\mathcal{EF} \neq \emptyset$, $\mathcal{EF}$ contains a part of the efficient frontier and $\mathcal{LB}$ is a lower bound set of the efficient frontier not contained in $\mathcal{EF}$

7.2. Benchmark description

7.2.1. Max-SAT-ONE

Let $\mathcal{F}$ be a boolean formula in conjunctive normal form. Max-SAT is the problem of finding a truth assignment such that the number of satisfied clauses in $\mathcal{F}$ is maximized. Max-ONE is the problem of finding a model for $\mathcal{F}$ with a maximum number of variables assigned to true. We define the Max-SAT-ONE problem as the problem of maximizing both the number of satisfied clauses and variables assigned to true. We experiment with the well-known dimacs SAT instances (Johnson and Trick, 1996). With current SAT solvers, these instances are solved almost instantly for the SAT problem. However, we will show that they remain quite challenging for the Max-SAT-ONE problem.

7.2.2. Biobjective vertex cover

Given a graph $G = (V, E)$, a *vertex cover* is a subset of vertices $S \subseteq V$ such that $\forall(u, v) \in E$, either $u \in S$ or $v \in S$. The *minimum vertex cover* is a vertex cover of minimum size. In the *weighted* version every vertex u has an associated weight $w(u)$ and the *weighted minimum vertex cover* is a vertex cover S with minimum $F(S) = \sum_{u \in S} w(u)$. In the biobjective version each vertex u has two weights $w_1(u)$ and $w_2(u)$ and the task is to minimize the two associated objective functions. In our experiments we generated random graph instances with parameters (N, E, C) where N is the number of vertices, E is the number of edges and C is the maximum weight. Instances are generated by randomly selecting E edges. For each vertex, two costs are randomly generated from the interval $[0 \ldots C]$.

7.2.3. Earth observation satellite scheduling

An earth observation satellite orbits the earth while taking photographs requested by different customers. Typically, it is impossible to fulfil all the requests. Thus, given a set of candidate photographs, the problem is to select the best subset that the satellite will actually take. The selected subset of photographs must satisfy a large number of imperative constraints and at the same time maximize the importance of selected photographs. Therefore, this is a mono-objective optimization problem. We experiment with instances from the Spot5 satellite (Bensana et al., 1999). When translated into the WCSP framework, these instances have unary, binary and ternary cost functions, and domains of size 2 and 4. We experiment with instances that include an additional capacity constraint imposed by the on-board storage limit. The scope of the capacity constraint is the whole set of variables. Therefore, the associated constraint graph is a clique and the induced width is the number of variables minus one. Clearly, mono-objective BE is an infeasible solving technique.

7.3. MO-BE for biobjective problems

In our first experiment we analyze the suitability of MO-BE in biobjective problems. To that end, we test MO-BE on the first and second benchmark (i.e., Max-SAT-ONE and biobjective minimum vertex cover). In Section 4, we showed that the applicability of MO-BE depends on the problem induced width. Therefore, we selected instances with induced width that we could handle with our computer (Pentium IV at 3GHz with 2 Gb of memory, running Linux). Variable orderings with small induced width were found with the min-degree heuristic (Dechter, 2003). Regarding Max-SAT-ONE, we found 35 dimacs instances with induced width below 24 which was our limit. Figure 6 compares the performance of MO-BE versus BID. The first, second and third columns contain the instances names, number of variables and number of clauses, respectively. The fourth column contains the induced width of each instance. The fifth column contains the efficient frontier size. The last two columns indicate the cpu time in seconds required by MO-BE and BID. A "-" indicates that the algorithm does not terminate in 30 minutes. It can be observed that, in accordance with the complexity analysis, the performance of MO-BE grows exponentially with the induced width. All the instances with small induced width (dubois and pret) are solved instantly. The aim instances, which have larger induced width, are still solved in less than half an hour. This experiment confirms once more that with current computers, it is the space and not the time what limits the applicability of decomposition methods. The BID presents a very bad performance in the dubois, pret and ssa instances, which indicates that this algorithm does not exploit the structure of the instances. BID can solve 7 out of the 10 aim instances, but it is generally slower than MO-BE. Only in instance aim-50-2-0-no-4 BID outperforms MO-BE.

For biobjective weighted minimum vertex cover, we generated samples of size 20 of the four classes of problems: $(60, 95, 4)$, $(70, 95, 4)$, $(80, 95, 4)$ and $(90, 95, 4)$. Figure 7 compares the performance of MO-BE versus BID. The first, second and third column indicate the problem class, the mean induced width and the mean size of the efficient frontier, respectively. The fourth and fifth columns report the mean cpu time required by MO-BE and BID. Observe that this instances have small induced width. Therefore MO-BE can solve them in a few seconds. However, BID can only solve the smaller class and requires two orders of magnitude more time than MO-BE.

					Time (sec.)	
Instance	nb. vars	nb. clauses	w^*	$\lvert\mathcal{EF}\rvert$	MO-BE	BID
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	-
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	-
aim-50-1_6-no-2	50	80	20	10	55	322
aim-50-1_6-no-3	50	80	19	10	15	-
aim-50-1_6-no-4	50	80	20	10	105	872
aim-50-1_6-yes1-1	50	80	18	10	23	654
aim-50-1_6-yes1-2	50	79	16	8	8	-
aim-50-1_6-yes1-3	50	80	19	10	65	362
aim-50-1_6-yes1-4	50	80	19	8	91	344
aim-50-2_0-no-4	50	98	23	10	1326	112
aim-50-2_0-yes1-3	50	100	23	14	1253	-
aim-50-2_0-yes1-4	50	100	21	14	297	954
ssa7552-158	1363	3034	11	329	1058.11	-
ssa7552-159	1363	3032	11	329	124.01	-

Fig. 6 Experimental results on Max-SAT-ONE problems. Each problem is solved with MO-BE and BID. Time limit 30 mins

Fig. 7 Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 4. Mean values on 20 instances for each parameter configuration. Time limit 30 mins

N	E			Time (sec.)	
(nb. vars)	(nb. edges)	w^*	$\lvert\mathcal{EF}\rvert$	MO-BE	BID
60	95	8.4	8.35	0.0665	715.7
70	95	7.7	8.5	0.025	-
80	95	6.55	7.85	0.0565	-
90	95	5.05	9.65	0.031	-

7.4. 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 compare MO-MBE and BID-lb on two domains: Max-SAT-ONE and biobjective weighted minimum vertex cover in problems that cannot be solved 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 covered by the lower bound set found by one method with respect the other one.

Figure 8 reports the results obtained for Max-SAT-ONE problems with high induce width. The sixth and eighth columns show the number of non-dominated points of the lower bound set found by MO-MBE (for each configuration of parameter k) and BID-lb with a time limit of 1800 seconds, respectively. The tenth column reports the ratio between the area covered by MO-MBE(k) for different values of k with respect to the area covered by BID-lb (i.e.,

Instance	nb.	nb.	w^*	MO-MBE(k)			BID-lb		
	vars	clauses		k	$\|\mathcal{EF}\|$	time (sec.)	$\|\mathcal{EF}\|$	time (sec.)	Ratio
aim-100-1-6-no-1	100	160	38	15 20	10 12	4.3 254.31	7	1800	1,5 2,19
aim-100-1-6-no-2	100	160	39	15 20	9 13	3.82 232.89	7	1800	1,15 2,68
aim-100-1-6-no-3	100	157	40	15 20	11 13	4.11 180.17	7	1800	1,75 2,72
aim-100-1-6-no-4	100	160	40	15 20	15 11	3.88 203.34	7	1800	3,28 1,75
aim-100-1-6-yes1-1	100	160	37	15 20	15 15	5.6 276.34	7	1800	3,31 3,47
aim-100-1-6-yes1-2	100	160	37	15 20	12 14	5.3 199.24	6	1800	2,57 3,34
aim-100-1-6-yes1-3	100	160	35	15 20	13 13	5.41 229.05	7	1800	2,59 2,47
aim-100-1-6-yes1-4	100	160	35	15 20	14 17	4.3 181.87	6	1800	3,62 5,27
aim-100-2-0-no-1	100	200	48	15 20	9 9	4.11 200.26	7	1800	1,34 4,41
aim-100-2-0-no-2	100	199	53	15 20	10 8	3.67 193.93	6	1800	1,77 1,19
aim-100-2-0-no-3	100	198	50	15 20	5 8	3.72 235.93	7	1800	0,41 0,97
aim-100-2-0-no-4	100	200	48	15 20	7 6	4.06 197.7	7	1800	0,75 0,5
aim-100-2-0-yes1-1	100	198	45	15 20	16 17	3.91 295.23	7	1800	4,06 5,19
aim-100-2-0-yes1-1	100	198	45	15 20	16 17	3.9 215.47	7	1800	4,06 5,19
aim-100-2-0-yes1-3	100	200	47	15 20	14 16	4.32 284.33	6	1800	3,27 4,93
aim-100-2-0-yes1-4	100	199	47	15 20	12 16	4.89 308.9	7	1800	2,25 4,5
aim-200-1-6-no-1	200	320	89	15 20	7 12	6.58 322.36	5	1800	1,26 3,84
aim-200-1-6-no-2	200	317	85	15 20	13 10	6.69 361.58	6	1800	3 1,77
aim-200-1-6-no-3	200	320	81	15 20	12 14	5.96 238.95	5	1800	3,47 5,11
aim-200-1-6-no-4	200	320	85	15 20	8 12	5.67 290.13	6	1800	1,19 2,77
aim-200-1-6-yes1-1	200	320	72	15 20	17 23	10.89 332.5	6	1800	5,71 10,75
aim-200-1-6-yes1-2	200	320	66	15 20	18 20	7.46 435.72	6	1800	5,54 6,76
aim-200-1-6-yes1-3	200	319	68	15 20	25 28	10.07 420.72	6	1800	11,69 14,69
aim-200-1-6-yes1-4	200	320	73	15 20	19 21	7.5 472.75	6	1800	6,81 8,57
aim-200-2-0-yes1-3	200	399	91	15 20	17 21	10.07 364.33	6	1800	6,09 8,92
ssa0432-003	435	1027	18	10 15	125 133	5.29 265.44	6	1800	501,95 570,85
ssa2670-130	1359	3321	27	10 15	342 349	54.33 1510.31	4	1800	7776,67 8343,78
ssa7552-038	1501	3575	29	5 10	254 357	15.65 128.32	4	1800	4185,67 8246,56
ssa7552-160	1391	3126	14	5 10	228 327	18.5 99.07	4	1800	3442,89 6845

Fig. 8 Experimental results on Max-SAT-ONE problems. Each problem is solved with MO-MBE(k) and BID-lb. Time limit 30 mins

Fig. 9 Experimental results on biobjective weighted minimum vertex cover problems. Parameter C is set to 4. Mean values on 20 instances for each parameter configuration. Time limit 30 mins

N	E		MO-MBE		
(nb. vars)	(nb. edges)	w^*	k	time (sec.)	Ratio
60	500	39.9	15	0.6	1.36
			20	31.95	1.74
			25	891.48	2.23
70	500	41.3	15	0.58	1.66
			20	38.22	2.20
			25	1040.63	2.36
80	500	45.6	15	0.8	2.60
			20	35.59	3.28
			25	965.49	3.54
90	500	47	15	1.17	3.22
			20	42.8	3.90
			25	1227.2	4.26
60	950	48.35	15	0.33	1.59
			20	12.37	2.14
			25	477.5	2.65
70	950	53.75	15	0.35	1.75
			20	15.53	2.47
			25	772.8	3.08
80	950	57.75	15	0.39	1.81
			20	25.9	2.78
			25	801.46	3.43
90	950	63.25	15	0.51	2.07
			20	23.44	2.96
			25	1043.77	3.82

area MO-MBE(k)/area BID-lb). For aim instances, the first thing to be observed is that the lowest value of k (i.e., $k = 15$) outperformes the approximations given by BID-lb for almost all instances. Note that the time spent by MO-MBE(15) is less than 8 seconds for all those instances while BID-lb reaches the time limit of 1800 seconds. Increasing the value of k allows MO-MBE to compute much more accurate lower bound sets and therefore, the ratio also increases. For ssa instances, the first thing to be noted is that the efficiency of BID-lb is very bad. MO-MBE is from 501.95 to 7776.67 times better than BID-lb with the lowest value of k. As before, the ratio increases with highest values of k.

For the second domain, we tested on samples of size 20 for the following parameter configurations ({60, 70, 80, 90}, {500, 950}, 4). Figure 9 reports the results obtained for MO-MBE for different values of the accuracy parameter k. The sixth column shows the ratio between the area covered by the lower bound set found by MO-MBE(k) and BID-lb with time limit 30 minutes. As can be observed, MO-MBE with the lowest value of k (i.e. $k = 15$) outperforms BID-lb for all parameter configurations. Note that BID-lb reaches the time limit for all instances, whereas the time spent by MO-MBE(15) is less than 2 seconds. When we fix the number of constraints and increase the number of variables, the efficiency of MO-MBE increases. When fixing the number of variables and increasing the number of constraints, the performance of MO-MBE decreases.

7.5. MO-MBE for mono-objective problems

In our third and final experiment we analyze the applicability of MO-MBE in mono-objective optimization problems with knapsack constraints. To that aim, we run MO-MBE on earth observation instances with capacity constraint. 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

				MO-MBE(k)		MBE(k)		ID	
Instance	nb. vars	nb. constr.	k	cost	time (sec.)	cost	time (sec.)	cost	time (sec.)
1506	940	14301	5	244433	0.31	123174	0.16	15612	1800
			10	249479	5.04	138216	1.36		
			15	254513	124.92	167267	25.74		
1401	488	10476	5	327152	0.25	125059	0.12	16084	1800
			10	333151	5.19	137060	0.97		
			15	343145	154.66	165064	21.57		
1403	665	12952	5	326249	0.3	121123	0.15	15774	1800
			10	339265	5.58	137131	1.22		
			15	340267	156.51	169144	26.3		
1405	855	17404	5	322426	0.38	117170	0.22	14782	1800
			10	334436	7.25	150171	1.69		
			15	341452	153.2	171195	35.58		
1407	1057	20730	5	321475	0.43	118172	0.28	14341	1800
			10	342519	6.67	147205	2.06		
			15	345543	281.7	175250	41.23		

Fig. 10 Experimental results on Spot5 instances. Each problem is solved with MO-MBE(k) and MBE(k) with different values of k, and iterative deepening with IlogSolver as solver engine. Time limit 1800 seconds

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. A third option to compute lower bounds in mono-objective problems is to use iterative deepening (ID) in the objective function. In this case, when the available time is exceeded, the algorithm returns a lower bound of the optimal value. The solver engine used for this algorithm is IlogSolver. Moreover, we model the capacity constraint as an IlogSolver's global constraint for improving its pruning capabilities.

We compare these three approaches. Figure 10 reports the lower bounds obtained for different values of 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 and ID. While for the first two approaches, 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.

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 a few knapsack-like constraints 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 constraints that can be processed with multiobjective algorithms. Finally, we want to evaluate the practical potential of branch and bound with MO-MBE for problems with unbounded induced width.

References

Bensana, E., M. Lemaitre, and G. Verfaillie. (1999). "Earth Observation Satellite Management." *Constraints* 4(3), 293–299.

Bertele, U. and F. Brioschi. (1972). *Nonserial Dynamic Programming*. Academic Press.

Bistarelli, S., H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. (1999). "Semiring-Based CSPs and Valued CSPs: Frameworks, Properties and Comparison." *Constraints* 4, 199–240.

Bistarelli, S., U. Montanari, and F. Rossi. (1997). "Semiring-Based Constraint Satisfaction and Optimization." *Journal of the ACM* 44(2), 201–236, March.

Brassard, G. and P. Bratley. (1995). *Fundamentals of Algorithms*. Prentice Hall, San Mateo, CA.

de Givry, S., G. Verfaillie, and T. Schiex. (1997). "Bounding the Optimum of Constraint Optimization Problems." In *Proc. of CP'97.*, Schloss Hagenberg (Austria). LNCS. Springer Verlag.

Dechter, R. (1999). "Bucket Elimination: A Unifying Framework for Reasoning." *Artificial Intelligence* 113, 41–85.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann, San Francisco.

Dechter, Rina and Irina, Rish. (2003). "Mini-Buckets: A General Scheme for Bounded Inference." *Journal of the ACM* 50(2), 107–153, March.

Ehrgott, M. and X. Gandibleux. (2001). "Bounds and Bound Sets for Biobjective Combinatorial Optimization Problems." *Lecture Notes in Economics and Mathematical Systems* 507, 241–253.

Ehrgott, M. and X. Gandibleux. (2002). *Multiple Criteria Optimization. State of the Art. Annotated Bibliographic Surveys.* Kluwer Academic Publishers.

Freuder, E.C. and R.J. Wallace. (1992). "Partial Constraint Satisfaction." *Artificial Intelligence* 58, 21–70, December.

Johnson, D.S. and M. Trick. (1996). "Second Dimacs Implementation Challenge: Cliques, Coloring and Satisfiability." *DIMACS Series in Discrete Mathematics and Theoretical Computer Science. AMS* 26.

Korf, R. (1985). "Depth-First Iterative-Deepening: An Optimal Admissable Tree search." *Artificial Intelligence* 27(3), 97–109.

Larrosa, Javier and Emma, Rollon. (2004a). "Adaptive Consistency with Capacity Constraints." In *Workshop on Modelling and Solving Problems with Constraints. ECAI'04.*

Larrosa, Javier and Emma, Rollon. (2004b). "Bucket Elimination with Capacity Constraints." In *6th Workshop on Preferences and Soft Constraints. CP'04.*

Meseguer, P., J. Larrosa, and M. Sanchez. (2001). "Lower Bounds for Non-Binary Constraint Optimization Problems." In *CP-2001*, pp. 317–331.

Neveu, Bertrand and Gilles, Trombettoni. (2003). "When Local Search Goes with the Winners." In *5th Int. Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems, CPAIOR 2003.*

Quimper, C., A. Lopez-Ortiz, P. van Beek, and A. Golynski. (2004). "Improved Algorithms for the Global Cardinality Constraint." In *Proc. of the 10th CP*, pages 542–556, Toronto (CA). LNCS 3258. Springer Verlag.

Regin, J.C. (1994). "A Filtering Algorithm for Constraints of Difference in CSPs." In *Proceedings of the 12th AAAI*, pp. 362–367.

Schiex, T., H. Fargier, and G. Verfaillie. (1995). "Valued Constraint Satisfaction Problems: Hard and Easy Problems." In *IJCAI-95*, pages 631–637, Montréal, Canada, August.

Sellmann, Meinolf. (2003). "Approximated Consistency for Knapsack Constraints." In *Proc. of the 9th CP*, pages 679–693. LNCS 2833. Springer Verlag.

Trick, Michael. (2003). "A Dynamic Programming Approach for Consistency and Propagation for Knapsack Constraints." *Annals of Operations Research* 118(118), 73–84.

van Hentenryck, P., H. Simonis, and M. Dincbas. (1992). "Constraint Satisfaction Using Constraint Logic Programming." *Artificial Intelligence* 58, 113–159.