An Algorithm for Multi-Criteria Optimization in CSPs

Marco Gavanelli[1]

Abstract. Constraint Satisfaction and Optimization are important areas of Artificial Intelligence. However, in many real-life applications, more functions should be optimized at the same time; the user needs to be provided a set of solutions and a posteriori choose the most preferable.

In this paper, we propose an algorithm for solving Multi-Criteria Optimization problems in this setting. The algorithm is complete, i.e., it finds all the non-dominated solutions, and does not make any assumption on the structure of the constraints nor on the type of the objective functions. It exploits Point Quad-Trees for the representation of the non-dominated frontier, in order to efficiently access the data. We describe the implementation and give experimental results showing that our algorithm outperforms widely used methods.

1 Introduction

Constraints are widely used in many fields, because they are declarative and efficiently solved. The usually addressed constraint problems fall into one of two frameworks: Constraint Satisfaction Problems (CSP) and the Constraint Optimization Problems (COP) [18]. A CSP consists of a set of variables ranging on given finite domains and subject to a set of constraints. A solution of a CSP is an assignment of values to variables satisfying all the imposed constraints. A COP is a CSP with an associated function that must be optimized. Very popular techniques are maintaining (Generalized) Arc Consistency (MAC) for solving CSPs and additional Branch and Bound (B&B) for COPs [18].

However, in real-life problems, the optimization function is not always clear. Often the user would like, ideally, to optimize two or more, possibly conflicting, criteria at the same time. In some cases, the criteria can be combined in a single function, e.g., by means of weighted sum, or distance from the ideal point. In other cases, the user cannot define a priori a combination, but would like to know various, possibly all, the tradeoff solutions, and choose a posteriori one of them. The user can be able to make better choices after knowing a variety of "good" (non subsumed) solutions, rather than defining a priori a combination, or a preference among solutions. The concept of multi-criteria optimization or Pareto optimality [10] was defined to address these problems. The feasible solutions are ranked by means of a partial order (instead of the total order induced by the optimization function): intuitively, a feasible solution S_1 is subsumed by another solution S_2 iff it gives worse (or equal) results in all the functions. As a result, the set of all non-dominated points in the CSP solution space are obtained. In this paper, we describe an efficient algorithm that provides the non-dominated points in a CSP; it is based on the concept of Optimization Nogoods and uses spatial data structures to arrange the set of nogoods.

[1] University of Ferrara, V. Saragat 1, 44100 Ferrara, Italy, email: mgavanelli@ing.unife.it

The rest of the paper is organized as follows. In Section 2 we give the basic definitions. In Section 3 we provide an algorithm that achieves the set of non-dominated solutions in a multi-criteria optimization problem. Section 4 provides some experimental results and Section 5 summarizes related works. Conclusions follow.

2 Definitions

We introduce some definitions used in the rest of the paper.

Definition 1 *A* Constraint Satisfaction Problem (CSP) *is a triple* $\langle X, D, C\rangle$ *where* $X = \{X_1, \ldots, X_n\}$ *is a set of variables,* $D = \{D_1, \ldots, D_n\}$ *is the set of domains and* C *is a set of constraints. An* Assignment $\mathcal{A} = \{X_i \mapsto v_i\}$ *is a function that maps some of the variables to values in the corresponding domain. A* Total assignment *involves all the variables. A* Feasible Solution *is a total assignment satisfying all the constraints.*

Definition 2 *A* Constraint Optimization Problem (COP) *is a quadruple* $\langle P, f, T, \leq\rangle$ *such that* P *is a CSP,* $\langle T, \leq\rangle$ *is a totally ordered set and* $f : D_1 \times \ldots \times D_n \mapsto T$ *is a function that maps each assignment to a value. An* Optimal Solution *is a feasible solution of* P *that maximizes the function* f.

With no lack of generality, we consider maximization problems.

For our needs, we will define the Multi-criteria Optimization Problem as follows:

Definition 3 *A* Multi-criteria Optimization Problem (MOP) *is a quadruple* $R = \langle P, \overline{f} = (f_1, \ldots, f_m), T, \leq\rangle$ *such that* P *is a CSP,* $f_1, \ldots, f_m$ *are functions* $f_i : D_1 \times \ldots \times D_n \mapsto T$ *and* $\langle T, \leq\rangle$ *is a totally ordered set.*

T^m is called the *Criterion Space*. In the rest of the paper, vectors in Criterion Space are indicated with an over line (e.g., $\overline{X}$).

Definition 4 *Given* $\overline{A}, \overline{B} \in T^m$*, we say that* $\overline{B}$ dominates $\overline{A}$*, (written* $\overline{A} \preceq \overline{B}$*) iff* $\forall_{i=1}^m A_i \leq B_i$.

Definition 5 *Given a MOP* $R = \langle P, \overline{f}, T, \leq\rangle$*, an assignment* S *is a* non-dominated solution *if it is a feasible solution of* P *and* $\nexists S'$ *solution of* P *such that* $\overline{f(S)} \preceq \overline{f(S')}$.

We will consider as solution of the MOP the set of non-dominated feasible assignments.

2.1 Algorithm van Wassenhove - Gelders

Usually, to find the non-dominated frontier, the optimal value is found in one direction and then the search is restarted constraining

the search space to better values for the other function. This algorithm has been widely used [5]; the first reference we are aware of is by van Wassenhove and Gelders [19]. The algorithm can be described as follows; note that it is applicable only to MOPs with two objective functions.

1. find the optimal solution wrt. function f_2; let us call it O_2. Let $\Delta \leftarrow O_2$
2. impose the constraint $f_1 > f_1(\Delta)$
3. maximize f_2. If a solution π exists, then it is non-dominated; else go to Step 5.
4. Let $\Delta \leftarrow f_1(\pi)$. Go to Step 2.
5. Stop.

3 An Algorithm for solving MOPs

Branch and Bound (B&B) [8] is an efficient, widely used method for solving COPs; it could be described as follows: first we find a solution (typically using a tree search), then we add a further constraint whose meaning is "new solutions must be *better* than the current best". Thus, the optimality problem is converted into a sequence of satisfiability problems. Operationally, the constraint can be propagated with bounds, obtained by solving a relaxation of the original problem. Of course this means keeping memory of the best solution achieved so far. Various implementations have been provided for this basic idea, for example in Constraint Logic Programming [11].

The optimization function can also be considered as a variable of the CSP, linked with other variables by means of a constraint[2]. Thus, the optimization constraint can be considered as a set of *nogoods*. A *nogood* is an assignment $\mathcal{A}$ such that there is no unenumerated solution containing $\mathcal{A}$. For example, if X and Y range in the set $\{0..4\}$, the function $f(X,Y) = X + Y$ can be represented with a variable F with domain $\{0..8\}$, linked with X and Y with the constraint $F = X+Y$. Once a feasible solution is found, e.g., $\{X \mapsto 1, Y \mapsto 0\}$, we learn that the optimal solution will have a greater value in F, so we can infer the nogoods $\{F \mapsto 0\}$ and $\{F \mapsto 1\}$.

For Multi-Criteria optimization, we can have a variable for each criterion (we call them *"criterion variables"*), and the inferred nogoods will involve more variables. For ease of presentation, we extend the concept of nogood as follows:

Definition 6 *An* Optimization Nogood *is a set of assignments* $\{F_1 \mapsto v_1, \ldots, F_k \mapsto v_k\}$ *such that* $\forall f_1 \leq v_1, \ldots, f_k \leq v_k$, $\{F_1 \mapsto f_1, \ldots, F_k \mapsto f_k\}$ *is a nogood.*

Intuitively, an optimization nogood prohibits dominated values for the criterion variables. In the previous example, since a solution with $F = 1$ was found, we can infer the optimization nogood $\{F \mapsto 1\}$. We now define how to learn optimization nogoods during a B&B search and how to exploit them to reduce the search space.

An optimization nogood can be inferred whenever a feasible solution is found: the value of all the objective functions $(F_1 \mapsto v_1, \ldots, F_m \mapsto v_m)$ is an optimization nogood. In fact, we are not interested in an assignment $\mathcal{A}'$ if a feasible assignment $\mathcal{A}$ that is better w.r.t. all the criteria exists. For example, if a feasible assignment $\mathcal{A}$ is known and it contains the bindings $F_1 \mapsto 1$ and $F_2 \mapsto 2$, we know that an assignment with $F_1 \mapsto 1$, $F_2 \mapsto 1$ is dominated.

Nogoods can be used to delete inconsistent values from the domains of the criterion variables. Each optimization nogood can be considered as an m-ary constraint that prohibits worse combinations of values for the criterion variables.

[2] Bounds can be employed with this implementation also.

We now show a generic tree-search algorithm that exploits optimization nogoods. We have a set AP of active problems, initially containing only the original CSP. While the set is not empty, we have an iteration. In each iteration a problem is selected from the set AP; if the selected problem can be shown unfeasible without search (line 5) (e.g., because one domain is empty), we do not consider it (it was simply removed from the set AC). If the problem can be directly solved (without search), and has only one solution Sol_p (line 6), then Sol_p is not dominated by any previous solution, and we record it in the set S. We also add the corresponding optimization nogood $ng(f_1(S_p), \ldots, f_m(S_p))$ as an m-ary constraint to all the active problems. If the selected problem cannot be directly solved, we branch (line 9), i.e., we generate the set of its children and add them to the active problems.

```
    let CSP = ⟨X, D, C⟩
    let {f_1, ..., f_m} be the criteria
 1: let AP = {CSP} % Set of Active Problems
 2: S ← ∅ % Set of Current Solutions
 3: While AP ≠ ∅
 4:   choose P ∈ AP, AP ← AP \ {P}
 5:   if inconsistency_detected(P) go to 3
 6:   if P has only one solution Sol_p
        % Add the optimization nogood to all the active problems
 7:     ∀Q ∈ AP, Q = ⟨X_q, D_q, C_q⟩,
        C_q ← C_q ∪ {ng(f_1(Sol_p), ..., f_m(Sol_p))}
      % remove from S the solutions subsumed by Sol_p and add Sol_p
 8:     S ← S \ {K ∈ S | f(K) ⪯ f(Sol_p)} ∪ {Sol_p}
      Else % Branch
 9:     Generate the set CP of children of P
10:     AP ← AP ∪ CP
11: End While
12: Return S
```

The space complexity of the algorithm, as for algorithm van Wassenhove and Gelders, is $O(|S|)$, in fact both algorithms record the whole set of nondominated solutions.

3.1 Organization of optimization nogoods

In many CLP(FD) systems [2, 1] each objective function can be represented as a domain variable, i.e., a variable with a domain. Considering $\overline{f} = (f_1, f_2, \ldots, f_m)$ the vector of all the objective functions, the domain of the vector function $\overline{f}$ is the cartesian product of the domains of the scalar functions: $D_{\overline{f}} = D_{f_1} \times D_{f_2} \times \ldots \times D_{f_m}$. Ideally, we would like to delete from the domain $D_{\overline{f}}$ all the area forbidden by the optimization nogoods; but we can only restrict the domains of the functions f_i, and $D_{\overline{f}}$ must always be the Cartesian product of the scalar domains. The following proposition provides a method to decide if a domain can be reduced by the set of optimization nogoods.

Proposition 1 *Let S be the set of previously found solutions, $\overline{f} = (f_1, f_2, \ldots, f_m)$ the objective function, D_{f_i} is the domain of f_i, $LUB_i = max(D_{f_i})$, $GLB_i = min(D_{f_i})$. Let us call $\overline{DP_i} \equiv (LUB_1, \ldots, LUB_{i-1}, GLB_i, LUB_{i+1}, \ldots, LUB_m)$; i.e., the vector which is "the best in all possible directions except direction i".*

The domain $D_{\overline{f}}$ can be reduced by the set of optimization nogoods only if $\exists i \in \{1..m\}$, $\exists s \in S$ such that $\overline{DP_i} \preceq \overline{f(s)}$.

Proof. *Suppose that the value v_i can be deleted from D_{f_i} by some optimization nogood. This means that all the values of $\overline{f}$ for which $f_i = v_i$ are dominated by that nogood.*

In particular, there will be a solution s such that $\overline{f(s)}$ dominates the point $(LUB_1, \ldots, LUB_{i-1}, v_i, LUB_{i+1}, \ldots, LUB_m)$ so, for transitivity, $\overline{DP_i} \preceq f(s)$.

Proposition 1 explains that in order to perform a reduction of the domains $D_{f_1}, \ldots, D_{f_m}$, we only need to check if the points $\overline{DP_1}, \ldots, \overline{DP_m}$ lie in the forbidden area. For example, consider Fig. 1. The gray area is forbidden by the optimization nogoods, represented as a set of non-dominated solutions. The domain $D_{\overline{f}}$ is the rectangle in the middle: it is the cartesian product of the domains D_{f_1} and D_{f_2}. Since we only consider reductions of the domains D_{f_1} and D_{f_2}, we can only delete the hatched area (in fact, the domain $D_{\overline{f}}$ must always be a rectangle). This means that we can reduce $D_{\overline{f}}$ only if the points $\overline{DP_1}$ and $\overline{DP_2}$ lie in the forbidden (gray) area.

From this observation follows that we do not have to check the current domain of $\overline{f}$ against all the optimization nogoods; if we are able to access the nogoods efficiently, we can have the same pruning by checking only m points. Since the nogoods are points in the criterion space, arranging them in a spatial data structure seems wise.

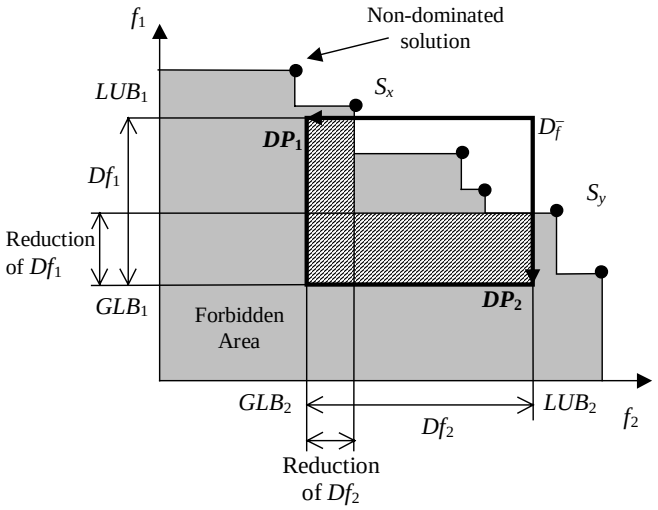

Figure 1. Region forbidden by the set of optimization nogoods

In our experiments, we adopted the Point Quad-Tree representation [16]. A Point Quad-Tree (in the following, simply Quad-Tree) contains points in a space; historically, it has been defined for two dimensions [3], but can be easily extended to any number of dimensions. For ease of presentation, we will give our examples in 2D, but all the following considerations can be extended to N dimensions unless stated otherwise. A Quad-Tree can be considered an adaptation of the binary search tree to more dimensions and it is based on the principle of recursive decomposition of the original space. Each element of the tree is either a leaf or a branch node. In two dimensions, each branch node is identified by a couple of coordinates (X, Y) and has four children (Fig. 2), representing the four quadrants in which the space is divided by the cartesian axes translated into the point (X, Y). The quadrants are normally labeled NW (for North-West), NE, SW and SE. In three dimensions, it is usually called Oct-Tree, because each node has eight children.

We believe that the Quad-Tree representation of the set of optimization nogoods is suitable for a series of reasons. (i) First of all, the Quad-Tree allows to access the data structure in $O(log|S|)$ time complexity, if S is the current set of non-dominated solutions; this

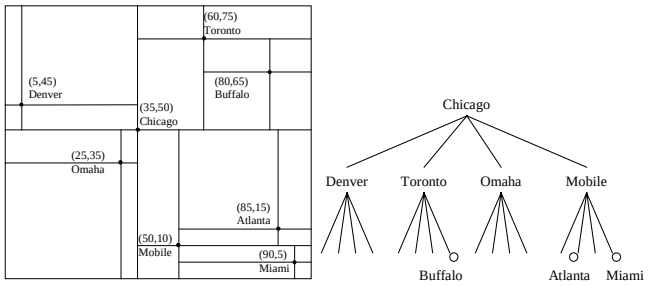

Figure 2. Example: records in a Quad-Tree

is very important because the check for domination is performed in every node of the tree. (ii) The *domination* between points is easily verifiable: the area dominated by a point in a Quad-Tree is simply the SW quadrant, while the area that dominates a point is the NE quadrant. So, if we store only non-dominated solutions, the SW quadrant of a node P is always empty, because it represents the area dominated by P. Also, the NE quadrant is empty as well, because if some points were in that quadrant, P would be dominated and could be removed. Thus, in two dimensions each node has only two children. (iii) The insertion of points can be obtained in $O(log|S|)$ time complexity.

As an example, in Fig. 3 we can see a Quad-Tree representation of the set of solutions in Fig. 1. The Quad-Tree corresponding to a set of points is not unique: it depends on the insertion order.

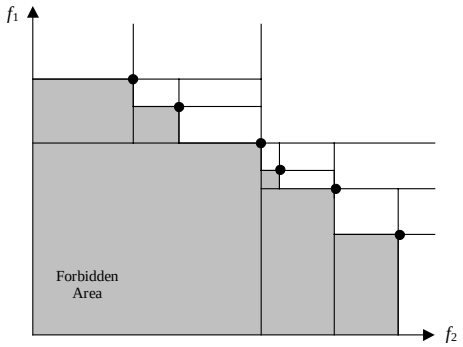

Figure 3. A Quad-Tree correspondent to a set of non-dominated solutions

One drawback of Quad-Trees is that deleting a point can be expensive, as it often requires the re-arrangement of the tree rooted in the deleted node. The algorithm by Samet [15] is often used; it consists of (i) finding a suitable new candidate as root of the subtree and (ii) re-inserting some children of the deleted node. In a way, we are "moving" the root to a new location; so, in order to maintain the tree consistent, we need to re-arrange the points that moved from a quadrant to another. For example, in Fig. 4 when we move the root from A to B we need to re-arrange the points in the gray area.

However, in our instance, a deletion occurs only if a better point is found, thus, a possible strategy is to substitute the old point with the new one. In this case, many of the points that should be re-arranged in the Samet algorithm [15] are dominated by the new solution, so they can be deleted as well. In Fig. 4, we need to delete the root A because a better point B was found. If we decide that the new root is B, we do not need to re-arrange the points in the gray area, because

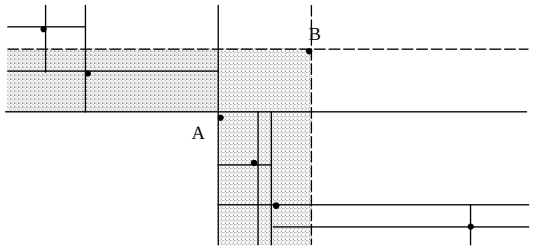

Figure 4. Point deletion in a Quad-Tree

they are dominated by B as well.

4 Experimental Results

We performed some experiments with a multi-knapsack problem and with randomly-generated problems. In the experiments, we used depth first search.

A knapsack problem consists of a set of items, with weight and profit associated to each item, and a total capacity of the knapsack. The task is to find a subset of items that fits into the knapsack and maximizes the profit [9]. The single-objective problem can be extended to multi-objective by allowing an arbitrary number of knapsacks [20]: given a set of o items and a set of m knapsacks, with

$p_{i,j}$ profit of item j according to knapsack i,
$w_{i,j}$ weight of item j according to knapsack i,
c_i capacity of knapsack i,

find a vector $\overline{x} = (x_1, x_2, \ldots, x_o) \in \{0,1\}^o$, such that $\forall i \in \{1, 2, \ldots, n\} : \sum_{j=1}^{o} w_{i,j} x_j \leq c_i$ and for which $\overline{f(x)} = (f_1(\overline{x}), f_2(\overline{x}), \ldots, f_m(\overline{x}))$ is maximum, where $f_i(\overline{x}) = \sum_{j=1}^{o} p_{i,j} x_j$ and $x_j = 1$ iff item j is selected.

We compared our algorithm with the algorithm by van Wassenhove and Gelders (Section 2.1); we can see the results on Table 1 (obtained with ECL^{i}PS^e [2] running on a 4 x UltraSPARC II 300 MHz with Solaris). Our algorithm performs better; moreover, the speedup is higher in difficult instances. Our method is also applicable to problems with more than two objective functions; in Table 2 timing results are given for problems with more criteria (obtained with ECL^{i}PS^e running on a Pentium II 400 MHz with Linux).

Table 1. Experimental results on Multi-Knapsack with 2 objective functions

Number of items	van Wassenhove-Gelders	Multi-B&B+Quad-tree
17	3.15 s	1.39 s
18	8.58 s	2.77 s
19	7.74 s	4.38 s
20	20.52 s	7.74 s
21	46.51 s	20.31 s
22	59.61 s	29.2 s
23	185.52 s	57.75 s
24	178.35 s	73.945 s
25	229.5 s	94.41 s
26	250 s	133.98 s
27	600.22 s	196.19 s
28	888.8 s	325.12 s
29	2133.42 s	659.23 s
30	3693.23 s	1207.17 s

Table 2. Experimental results on Multi-Knapsack with more objective functions

Number of Items	2D	3D	4D
10	0.04 s	0.09 s	0.14 s
11	0.06 s	0.11 s	0.33 s
12	0.09 s	0.2 s	0.46 s
13	0.13 s	0.33 s	0.97 s
14	0.21 s	0.68 s	1.76 s
15	0.34 s	1.62 s	3.45 s
16	0.4 s	2.25 s	9.11 s
17	0.79 s	5.67 s	10.31 s
18	1.8 s	7.9 s	24.72 s
19	2.45 s	16.98 s	53.47 s
20	4.7 s	24.06 s	144.97 s
21	11.37 s	57.04 s	286.65 s
22	18.07 s	63.31 s	507.9 s

We also performed some tests on random CSPs, generated as proposed in [13]. A CSP is generated given four parameters: the number n of variables, the size d of each domain, the probability p that a constraint exists on a couple of variables and the conditional probability q that a couple of assignments are inconsistent given that the variables are linked by a constraint. For simplicity, we randomly generated linear objective functions. In the graph in Fig. 5, each bar represents the average of ten problems, with $n = 10$, $d = 15$, constraint density p ranging from 20 to 100%, and tightness q from 10 to 90%.

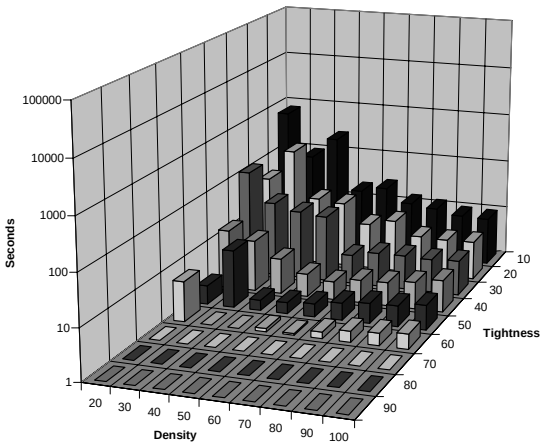

Figure 5. Performance of Multi-B&B on randomly-generated MOPs

In Fig. 6 the ratio of the timing results for the same problems is shown. Our algorithm is up to 18 times faster than the algorithm by van Wassenhove-Gelders, while it is never worse than 32%. It is worth noting that the instances in which our algorithm is slower than van Wassenhove-Gelders are usually easy instances. For example, the worst ratio is 0.75, which is due to problems solved on average in 0.082 seconds by Multi-B&B and in 0.062 by van Wassenhove-Gelders. On the other hand, one of the best ratios (nearly 18), was obtained in a class of problems where van Wassenhove-Gelders took on average 13067 seconds (about 3 hours and a half) while our method was able to find the non-dominated frontier in about 12 minutes.

5 Related work

Usually, multi-criteria problems are addressed by translating the problem into one or more problems with a single objective, each

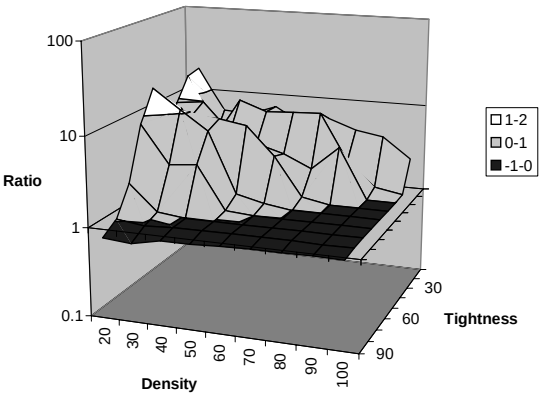

Figure 6. Ratio of computation time of WG and Multi-B&B

solved through standard single objective algorithms. Some methods convert the MOP into a COP. The *Weighting Objective Method* [17] considers a linear combination of the objective functions, with positive coefficients that sum up to one. The coefficients are assumed beforehand, and they can be varied to obtain other solutions. However, if the problem's criterion space (i.e., the image through the objective functions of the set of solutions) is not convex, there is no guarantee that all the nondominated points can be generated.

The *Hierarchical Method* and *Trade-Off Method* [17] rank the objective functions in order of importance. A solution S' is found optimizing only the most important function (say f_1), then function f_1 is converted into a constraint; the obtained problem is then solved optimizing with respect to function f_2 and so on.

The *Global Criterion Method* [14] tries to minimize a distance from the ideal solution. The ideal solution is computed by solving all the COPs with only one objective function. The ideal value for the function f_i is obtained by solving the problem with only f_i as objective function; in this way an ideal vector $S^0 = (s_1^0, \ldots, s_m^0)$ is obtained. Now a new COP is solved whose objective function is $f(x) = \sum_{i=1}^{m} \left(\frac{s_i^0 - f_i(x)}{s_i^0} \right)^p$, where p defins the type of distance.

In *Goal Programming* [17] [7] the objectives are conceptualized as goals, then priorities or weights are defined for goals. Now, deviational variables are defined to measure how much a given goal has been achieved. For instance, for an objective $max(f_i(x))$ (i.e., maximize $f_i(x)$) a goal is defined: it can be a realistic or utopian value t_i. Then the goal is changed into two constraints stating $f_i(x) - d_i^- = t_i$, $d_i^- \geq 0$. We obtain a problem with more variables, where we have to minimize all the d_i; this problem is usually addressed by weighting factors or priorities, falling thus in one of the previous categories.

Note that all these methods do not provide the whole efficient frontier (the set of non dominated points), but they only find one solution that will hopefully satisfy the user. Also, *Interactive methods* (that try to interact with the user to find a solution that he/she finds acceptable) have been described [17].

Various methods have been developed for the linear case [12] (where constraints are linear), but they are beyond the scope of this work. The interested reader can refer to [17].

It is worth noting that all these methods try to translate a MOP into one or more COPs; they do not address directly the MOP problem. Other methods address the MOP but are incomplete, exploiting genetic algorithms [4] or tabu search [6].

6 Conclusions

We presented an algorithm that extends Branch-and-Bound for Multi-criteria Optimization in CLP(FD). The algorithm is complete, meaning that it finds the whole non-dominated frontier. It is applicable to any CSP, as it does not make any assumption on the structure of constraints, nor on the type of objective functions. It infers nogoods for each feasible solution found and it uses Point Quad-Trees to efficiently store the set of nogoods. We compared the algorithm with a widely used method and it resulted more efficient, particularly in the difficult instances, both in multi-knapsack problems and in randomly-generated problems.

ACKNOWLEDGEMENTS

I would like to thank Carmen Gervet and Michela Milano for very useful suggestions on this work. I also wish to thank the anonymous referees for their comments.

REFERENCES

[1] M. Carlsson, J. Widén, J. Andersson, S. Andersson, K. Boortz, H. Nilsson, and T. Sjöland, 'SICStus prolog user's manual', Technical Report T91:15, Swedish Institute of Computer Science, (June 1995).
[2] ECRC and IC-Parc, *ECL^{i}PS^e User Manual, Release 5.2*, IC-Parc, Imperial College, London, UK, 2001.
[3] R.A. Finkel and J.L. Bentley, 'Quad trees: A data structure for retrieval on composite keys', *Acta Informatica*, **4**(1), 1–9, (November 1974).
[4] C.M. Fonseca and P.J. Fleming, 'An Overview of Evolutionary Algorithms in Multiobjective Optimization', *Evolutionary Computation*, **3**(1), 1–16, (Spring 1995).
[5] C. Gervet, Y. Caseau, and D. Montaut, 'On refining ill-defined constraint problems: A case study in iterative prototyping', in *PACLP-99*, pp. 255–275, London, (1999).
[6] M.P. Hansen, 'Tabu search for multiobjective optimization: MOTS', in *13th International Conference on Multiple Criteria Decision Making (MCDM'97)*, Cape Town, South Africa, (January 1997).
[7] Y. Ijiri, *Management Goals and Accounting for Control*, North Holland, Chicago, 1965.
[8] E.L. Lawler and D.E. Wood, 'Branch-and-bound methods: a survey', *Operations Research*, **14(4)**, 699–719, (1966).
[9] S. Martello and P. Toth, *Knapsack problems: algorithms and computer implementations*, John Wiley & Sons, 1990.
[10] V. Pareto, *Cours d'Economie Politique*, F. Rouge, Lausanne, 1896.
[11] S. Prestwich, 'Three implementations of branch-and-bound in CLP', in *Proceedings of Fourth Compulog-Net Workshop on Parallelism and Implementation Technologies*, Bonn, (September 1996).
[12] M.J. Rosenblatt and Z. Sinuany-Stern, 'Generating the discrete efficient frontier to the capital budgeting problem', *Operations Research*, **37**(3), 384 – 394, (May - June 1989).
[13] D. Sabin and E.C. Freuder, 'Contradicting Conventional Wisdom in Constraint Satisfaction', in *Proc. of PPCP'94*, volume 874 of *Lecture Notes in Computer Science*, pp. 10–20, (May 1994).
[14] M.E. Salukvadze, 'On the existence of solution in problems of optimization under vector valued criteria', *Journal of Optimization Theory and Applications*, **12**(2), 203–217, (1974).
[15] H. Samet, 'Deletion in two-dimensional quad trees', *Communications of the ACM*, **23**(12), 703–710, (December 1980).
[16] H. Samet, 'The quadtree and related hierarchical data structures', *ACM Computing Surveys*, **16**(2), 187–260, (June 1984).
[17] R.E. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application*, Wiley, New York, 1986.
[18] E.P.K. Tsang, *Foundation of Constraint Satisfaction*, Academic Press, 1993.
[19] L.N. Van Wassenhove and L.F. Gelders, 'Solving a bicriterion scheduling problem', *European Journal of Operational Research*, **4**(1), 42–48, (1980).
[20] E. Zitzler and L. Thiele, 'Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach', *IEEE Transactions on Evolutionary Computation*, **3**(4), 257–271, (1999).