# Cost-Based Domain Filtering

F. Focacci[2], A. Lodi[1], and M. Milano[2]

[1] DEIS, Univ. Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
alodi@deis.unibo.it
[2] Dip. Ingegneria, Univ. Ferrara, Via Saragat, 41100 Ferrara, Italy
{ffocacci,mmilano}@deis.unibo.it

**Abstract.** Constraint propagation is aimed at removing from variable domains combinations of values which cannot appear in any consistent solution. Pruning derives from *feasibility reasoning*. When coping with optimization problems, pruning can be performed also on the basis of costs, i.e., *optimality reasoning*. Propagation can be aimed at removing combination of values which cannot lead to solutions whose cost is better then the best one found so far. For this purpose, we embed in global constraints optimization components representing suitable relaxations of the constraint itself. These components provide efficient Operations Research algorithms computing the optimal solution of the relaxed problem and a gradient function representing the estimated cost of each variable-value assignment. We exploit these pieces of information for pruning and for guiding the search. We have applied these techniques to a couple of ILOG Solver global constraints (a constraint of difference and a path constraint) and tested the approach on a variety of combinatorial optimization problems such as Timetabling, Travelling Salesman Problems and Scheduling Problems with setup. Comparisons with pure Constraint Programming approaches and related literature clearly show the benefits of the proposed approach. By using cost-based filtering in global constraints, we can optimally solve problems that are one order of magnitude greater than those solved by pure CP approaches, and we outperform other hybrid approaches integrating OR techniques in Constraint Programming.

## 1 Introduction

Finite Domain Constraint Programming (CP) has been recognized as a powerful tool for modelling and solving combinatorial optimization problems. CP tools provide *global constraints* offering concise and declarative modelling capabilities together with efficient and powerful *domain filtering* algorithms. These algorithms remove combinations of values which cannot appear in any consistent solution.

When coping with optimization problems, an objective function $f$ is defined on problem variables. With no loss of generality, we restrict our discussion to minimization problems. CP systems usually implement a Branch and Bound algorithm to find an optimal solution. The idea is to solve a set of satisfiability

J. Jaffar (Ed.): CP'99, LNCS 1713, pp. 189–203, 1999.

problems (i.e., a feasible solution is found if it exists), leading to successively better solutions. In particular, each time a feasible solution $s^*$ is found (whose cost is $f(s^*)$), a constraint $f(x) < f(s^*)$ is added to each subproblem in the remaining search tree. The purpose of the added constraint, called *upper bounding constraint*, is to remove portions of the search space which cannot lead to better solutions than the best one found so far. The problem with this approach is twofold: (*i*) only the upper bounding constraint is used to reduce the domain of the objective function; (*ii*) in general, the link between the variable representing the objective function and problem decision variables is quite poor and does not produce effective domain filtering.

As concerns the first point, previous works have been proposed that compute also lower bounds on the objective function by (possibly optimally) solving relaxed problems [2,5], [21,22].

Concerning the second point, two notable works by Caseau and Laburthe ([6] and [7]) embed in optimization constraints lower bounds from Operations Research and define a regret function used as heuristic information. Here we propose a further step in the integration of OR technology in CP, by using well known OR techniques, i.e., lower bound calculation and reduced cost fixing [15], for cost-based propagation. We embed in global constraints an *optimization* component, representing a proper relaxation of the constraint itself. This component provides three information: (*i*) the optimal solution of the relaxed problem, (*ii*) the optimal value of this solution representing a lower bound on the original problem objective function, and (*iii*) a *gradient function* $grad(V, v)$ which returns, for each possible couple variable-value $(V, v)$, an optimistic evaluation of the additional cost to be paid if $v$ is assigned to $V$. The *gradient function* extends and refines the notion of regret used in [6] and [7]. We exploit these pieces of information both for propagation purposes and for guiding the search.

We have implemented this approach on two global constraints in ILOG Solver [18]: a constraint of difference and a path constraint. The *optimization* component used in both constraints embeds the Hungarian Algorithm [4] for solving Assignment Problem (AP) which is a relaxation of the problem represented by the path constraint and exactly the same problem as the one modelled by the constraint of difference. The Hungarian Algorithm provides the optimal solution of the AP, its cost and the *gradient function* in terms of reduced costs matrix. Reduced costs provide a significant information allowing to perform cost-based domain filtering, and to guide the search as heuristics. In general however, any relaxation can be used, e.g., a LP relaxation or a spanning tree (spanning forest) for the path constraint, provided that it produces the information needed (i.e., the lower bound and reduced costs).

We have used the resulting constraints to solve Timetabling Problems, Travelling Salesman Problems and Scheduling Problems with setup times (where the path constraint has been interpreted and adapted to be a multi-resource transition time constraint). By using the cost-based domain filtering technique in global constraint, we achieve a significant computational speedup with respect to traditional CP approaches: in fact, we can optimally solve (and prove optimal-

ity for) problems which are one order of magnitude greater than those solved by pure CP approaches. Also, comparisons with related literature describing other OR-based hybrid techniques show that integrating cost-based reduction rules in global constraints gets unarguable advantages.

## 2 Motivations and Background

In this section, we present the main motivation of this paper. We start from the general framework, Branch & Infer, proposed by Bockmayr and Kasper [3], which unifies and subsumes Integer Linear Programming (ILP) and Constraint Programming (CP). In a constraint language, the authors recognize two kind of constraints: *primitive* and *non primitive* ones. Roughly speaking, primitive constraints are those which are easily handled by the constraint solver, while non primitive ones are those for which it does not exist a (complete) method for satisfiability, entailment and optimization running in polynomial time. Thus, the purpose of a computation in a constraint-based system is to infer primitive constraints $p$ from non primitive ones $c$.

As mentioned, when solving optimization problems, CP systems usually perform the branch and bound method. In particular, each time a feasible solution $s^*$ is found (whose cost is $f(s^*)$), a constraint $f(x) < f(s^*)$ is added to each subproblem in the remaining search tree. The purpose of the added *upper bounding constraint* is to remove portions of the search tree which cannot lead to better solution than the best one found so far. Two are the main limitations of this approach: $(i)$ we do not have good information on the problem lower bound, and consequently, on the quality of the solutions found; $(ii)$ the relation between the cost of the solution and the problem variables is in general not very tight, in the sense that is usually represented by a non primitive constraints.

Many works have been proposed in order to solve the first problem by computing a lower bound on the problem, thus obtaining in CP a behaviour similar to the OR branch and bound technique. In global constraints, for example, a lower bound is computed on the basis of variable bounds involved in the constraint itself, see for instance [22]. Alternatively, Linear Programming (LP) [8] can be used for this purpose as done for example in [2,5,21].

The second problem arises from the fact that in classical CP systems primitive constraints are the following:

$$Prim = \{X \leq u, X \geq b, X \neq v, X = Y, integral(X)\}$$

where $X$ and $Y$ are variables, $u$, $v$, $b$ are constants. All other constraints are *non primitive*. The branch and bound *a-la* CP would be very effective if the *upper bounding constraint* would be a primitive constraint. Unfortunately, in general, while the term $f(s^*)$ is indeed a constant, the function $f(x)$ is in general not efficiently handled by the underlying solver.

For example, in scheduling problems, the objective function may be the *makespan* which is computed as the $max_{i \in Task}\{St_i + d_i\}$ where $St_i$ is a variable representing the start time of Task $i$ and $d_i$ its duration. In matching, timetabling

and travelling salesman problems, each variable assignment is associated with a cost (or a penalty), the objective function is the sum of the assignment costs. In these cases, the function $f$ representing the objective function makes the upper bounding constraint a non primitive one.

The general idea we propose is to infer primitive constraints on the basis of information on costs. We use *optimization* components within global constraints representing a proper relaxation of the problem (or exactly the same problem) represented by the global constraint itself. The optimization component provides the optimal solution of the relaxed problem, its value and a gradient function computing the cost to be added to the optimal solution for each variable-value assignment. In this section, we provide an intuition on how this information is exploited. In section 3 we formally explain the proposed technique.

With no loss of generality, we consider here as optimization component a Linear Program (LP) representing a (continuous) linear relaxation of the constraint itself. The optimal solution of the relaxed problem can be used as heuristic information as explained in section 4. The optimal value of this solution improves the lower bound of the objective function and prunes portions of the search space whose lower bound is bigger than the best solution found so far. The reduced costs associated to linear variables is proportional to the cost to be added to the optimal solution of the relaxed problem if the corresponding linear variable becomes part of a solution. If this sum is greater than the best solution found so far, the linear variable can be fixed to 0, i.e., it is excluded from the solution. This technique is known in OR as variable fixing [15]. Given a mapping between LP and CP variables, we have the same information for CP variable domain values. Thus, we can infer primitive constraints of the kind $X \neq v$, and we prune the subproblem defined by the branching constraint $p = (X = v)$.

The advantage of this approach is twofold. First, we exploit cost-based information for domain filtering in global constraints. The advantage with respect to traditional OR variable fixing technique is that in our case domain filtering usually triggers propagation of other constraints thanks to shared variables. Second, we do not need to define each time a proper relaxation of the original problem, but we associate a proper relaxation to each global constraint which can be written once for all for optimization purposes. A complementary approach could instead generate a single linear program containing a linearization of the inequalities corresponding to the whole set of constraint representing the problem as done in [21]. This would allow to have one single global optimization constraint in the form of LP. However, it can be applied only if we consider as a relaxed problem a linear problem, while our approach is more general and we can apply more sophisticated techniques such as additive bounds [10].

## 3 Global Optimization Constraints

In this paper we apply our ideas on two global constraints of ILOG solver: a constraint of difference (`IlcAllDiff`) and a path constraint (`IlcPath`) which was extended in order to handle transition costs depending on the selected path.

The constraint `IlcAllDiff` [19] applied to an array of domain variables $Vars = (X_1, ..., X_n)$, ensures that all variables in $Vars$ have a different value.

The constraint `IlcPath` ensures that, given a set of nodes $I$, a maximum number of paths `NbPath`, a set of starting nodes $S$ and a set of ending nodes $E$, there exists at most `NbPath` paths starting from nodes in $S$, visiting all nodes in $I$ and ending at nodes in $E$. Each node will be visited only once, will have only one predecessor and only one successor. The constraint works on an array of domain variables $Next$, each representing the next node in the path ($Next[i] = j$ if and only if node $i$ precedes $j$ in the solution).

In both cases, as LP relaxation we use the Assignment Problem (AP) solved by the Hungarian algorithm described in [4]. We have chosen the AP solver as a Linear Component for two reasons: $(i)$ it is a suitable relaxation for the `IlcPath` constraint and exactly the same problem represented by `IlcAllDiff` constraint; $(ii)$ we have a specialized, polynomial and incremental algorithm (the Hungarian method) for solving it and computing the reduced costs[1]. Notice that the proposed approach is independent from the used relaxation. In fact, the algorithm providing lower bound values and reduced costs can be seen as a software component, and it can be easily substituted by other algorithms. For example, an algorithm which incrementally solves the Minimum Spanning Arborescence can be easily used instead of the Hungarian algorithm for computing the lower bound and the reduced costs for the path constraint as shown in [13].

Two important points that should be defined are $(i)$ the mapping between variables appearing in the global constraint and variables appearing in the AP formulation; $(ii)$ the cost based propagation.

In the next sections, we formally define the Assignment Problem, the mapping and the cost-based propagation.

### 3.1 The Assignment Problem as *Optimization* Component

The well known *Linear Assignment Problem* (AP) (see [9] for a survey) states as follows. Given a square cost matrix $c_{ij}$ of order $n$, the problem is to assign to each row a different column, and vice versa in order to minimize the total sum of the row-column assignment costs.

This problem can be seen as the *Minimum Cost Perfect Matching* problem. Let $G = (V \cup T, A)$ be a bipartite graph where $V$ and $T$ are the vertex sets and $|V| = |T| = n$, $A = \{(i, j)|i \in V, j \in T\}$ the arc set, and $c_{ij}$ is the cost of arc $(i, j) \in A$. The minimum cost perfect matching gives the solution to the AP. Vertex $i \in V$ corresponds to row $i$ and vertex $j \in T$ to column $j$. A classic Integer Linear Programming (ILP) formulation for the AP is:

$$Z(AP) \quad = \min \sum_{i \in V} \sum_{j \in T} c_{ij}\, x_{ij} \tag{1}$$

[1] Note that the AP can be formulated as an Integer Linear Program. However, being the cost matrix totally unimodular, the LP relaxation of the AP always provides an integer (thus optimal) solution.

$$\text{subject to} \sum_{i \in V} x_{ij} = 1, \qquad j \in T \tag{2}$$

$$\sum_{j \in T} x_{ij} = 1, \qquad i \in V \tag{3}$$

$$x_{ij} \text{ integer}, \quad i \in V, j \in T \tag{4}$$

where $x_{ij} = 1$ if and only if arc $(i, j)$ is in the optimal solution. Constraints (2) and (3) impose in-degree and out-degree of each vertex equal to one.

Alternatively, AP can also be defined on a digraph (of $n$ vertices) as the graph theory problem of finding a set of *disjoint* sub-tours such that all the vertices in the digraph are visited and the sum of the costs of selected arcs is a minimum.

It is well-known that the AP optimal solution can be obtained through a *primal-dual* algorithm. We have used a C++ adaptation of the Hungarian algorithm described in [4]. The solution of the AP requires in the worst case $O(n^3)$, whereas each re-computation of the optimal AP solution, needed in the case of modification of one value in the cost matrix, can be efficiently computed in $O(n^2)$ time through a single augmenting path step.

The information provided by the Hungarian algorithm is the AP optimal solution and a reduced cost matrix $\bar{c}$. In particular, for each arc $(i, j) \in A$ the reduced cost value is defined as $\bar{c}_{ij} = c_{ij} - u_i - v_j$, where $u_i$ and $v_j$ are the optimal values of the Linear Programming dual variables associated with the $i$-th constraint of type (2) and the $j$-th constraint of type (3), respectively. The reduced cost values are obtained from the AP algorithm without extra computational effort during AP solution. Each $\bar{c}_{ij}$ is a lower bound on the cost to be added to the optimal AP solution if we force arc $(i, j)$ in solution.

### 3.2 Mapping

In this section, we define the mapping between variables and constraints used in our optimization component and those used in the CP program. The mapping between the ILP formulation and the CP formulation is straightforward and has been previously suggested in [21]. In CP, we have global constraints involving variables $X_1, ..., X_n$ (in the path constraints they are called $Next_i$), ranging on domains $D_1, ..., D_n$, and cost $c_{ij}$ of assigning value $j \in D_i$ to $X_i$. Obviously, the cost of each value not belonging to a variable domain is infinite. The problem we want to solve is to find an assignment of values to variables consistent with the global constraint, and whose total cost is minimal. If an ILP variable $x_{ij}$ is equal to 1, the CP variable $X_i$ is assigned to the value $j$, $x_{ij} = 1 \leftrightarrow X_i = j$. Constraints (2) and (3) correspond to a constraint of difference imposing that all CP variables assume different values. The ILP objective function corresponds to the CP objective function.

It is worth noting that the AP codes work on square matrices, while, in general, in the CP problem considered, it is not always true that the number of variables is equal to the number of values. Thus, the cost matrix of the original problem should be changed. Suppose we have $n$ variables $X_1, \ldots, X_n$, and suppose that the union of their domains contains $m$ different values. A necessary

condition for the problem to be solvable is that $m \geq n$. The original cost matrix has $n$ rows (corresponding to variables) and $m$ columns (corresponding to values). Each matrix element $c_{ij}$ represents a cost of assigning $j$ to $X_i$ if value $j$ belongs to the domain of $X_i$. Otherwise, $c_{ij} = +INF$. In addition, we have to change the matrix so as to have a number of rows equal to the number of columns. Thus, we can add to the matrix $m - n$ rows where each value $c_{ij} = 0$ for all $i = n+1, \ldots, m$ and for all $j = 1, \ldots, m$, obtaining an $m \times m$ cost matrix. The addition of these $m - n$ rows brings the algorithm to a time complexity of $O(mn^2)$ (and not $O(m^3)$), whereas each re-computation of the optimal AP solution requires only $O(nm)$ time.

Note that the constraint of difference and the AP component have exactly the same semantics: they compute a solution where all variables are assigned to different values. Thus, each solution of the AP is feasible for the constraint of difference. In general, in a CP program the same variables appear in different constraints. Thus, the constraint of difference alone (and the AP component alone) can be seen as a relaxation of a more general problem. As a consequence, the AP optimal solution $Z_{LB}$ is a lower bound on the optimal solution of the overall problem. On the contrary, when used within a path constraint, the AP component represents a relaxation of the constraint itself (where sub-tours may appear) and it is no longer true that the optimal solution of the AP is feasible for the path constraint. In this case, the AP optimal solution $Z_{LB}$ is a lower bound of the sum of the arcs appearing in the path constraint.

As already mentioned, the AP provides a reduced cost matrix. Given the mapping between LP and CP variables, we know that the LP variable $x_{ij}$ corresponds to the value $j$ in the domain of the CP variable $X_i$. Thus, the reduced cost matrix $\bar{c}_{ij}$ provides information on CP variable domain values, $grad(X_i, j) = \bar{c}_{ij}$.

### 3.3 The Cost-Based Propagation

In this section we describe filtering techniques based on the information provided by the optimization component. We have a first (trivial) propagation based on the AP optimal value $Z_{LB}$. At each node of the search tree, we check the constraint $Z_{LB} < Z$ where $Z$ is the variable representing the CP objective function. This kind of propagation generates a yes/no answer on the feasibility of the current node of the search tree; therefore it does not allow any real interaction with the other constraints of the problem.

More interesting is the second propagation from reduced costs $\bar{c}$ towards decision variables $X_1, ..., X_n$, referred to as RC-based propagation. This filtering algorithm directly prunes decision variables $X_1, ..., X_n$ domains on the basis of reduced costs $\bar{c}$. Suppose we have already found a solution whose cost is $Z^*$. For each domain value $j$ of each variable $X_i$, $Z_{LB_{X_i=j}} = Z_{LB} + \bar{c}_{ij}$ is a lower bound value of the subproblem generated if value $j$ is assigned to $X_i$. If $Z_{LB_{X_i=j}}$ is greater or equal to $Z^*$, $j$ can be deleted from the domain of $X_i$. This filtering algorithm performs a real back-propagation from $Z$ to $X_i$. Such domain filtering usually triggers other constraints imposed on shared variables, and it appears therefore particularly suited for CP. Indeed, the technique proposed represents

a new way of inferring primitive constraints starting from non primitive ones. In particular, primitive constraints added (of the form $X_i \neq j$) do not derive, as in general happens, from reasoning on feasibility, but they derive from reasoning on optimality. Furthermore, note that the same constraints of the form $X_i \neq j$ are also inferred in standard OR frameworks (variable fixing). However, this fixing is usually not exploited to trigger other constraints, but only in the next lower bound computation, i.e., the next branching node.

When the AP is used as optimization component, an improvement on the use of the reduced costs can be exploited as follows: we want to evaluate if value $j$ could be removed from the domain of variable $X_i$ on the basis of its estimated cost. Let $X_i = k$ and $X_l = j$ in the optimal AP solution. In order to assign $X_i = j$, a minimum augmenting path, say PTH, from $l$ to $k$ has to be determined since $l$ and $k$ must be re-assigned. Thus, the cost of the optimal AP solution where $X_i = j$ is $Z_{LB} + \bar{c}_{ij} + cost(PTH)$, by indicating with $cost(PTH)$ the cost of the minimum augmenting path PTH. In [13], two bounds on this cost have been proposed, whose calculation does not increase the total time complexity of the filtering algorithm ($O(n^2)$). We will refer to this propagation as *improved reduced cost* propagation (*IRC-based* propagation).

### 3.4 Propagation Events

In this section, we describe the data structures which should be built and maintained by the global constraints, and the events triggering propagation.

When the constraint is stated for the first time, the cost matrix is built and the Hungarian Algorithm is used to compute the AP optimal solution and the reduced cost matrix in $O(n^3)$. Each time the AP optimal solution is computed, the lower bound of the variable representing the objective function is updated and the RC-based propagation is performed (or IRC-based if the corresponding flag is set). The constraint is triggered each time a change in a variable domain happens and each time the upper bound of the objective function is updated. Each time a value $j$ is removed from the domain of variable $X_i$, the cost matrix is updated by imposing $c_{ij} = +\infty$, i.e., $x_{ij} = 0$. If value $j$ belongs to the solution of the AP (and only in this case), the lower bound $Z_{LB}$ is updated by incrementally re-computing the assignment problem solution in $O(n^2)$. The AP re-computation leads to a new reduced cost matrix. Thus, the RC-based propagation (or IRC-based) is triggered and some other values may be removed.

Note that since the re-computation of the AP solution is needed only if the value removed from the domain of a variable is part of the current AP solution, it is possible to write the optimization constraint in such a way that whenever a value is assigned to a variable only one incremental re-computation is needed. Each time the objective function upper bound is updated, the RC-based propagation (or IRC-based) is triggered.

## 4 Heuristics

The optimal solution of a relaxed problem, the lower bound value, and the set of reduced costs can be used for the heuristics during the search for a solution. Different examples of such use are described in the next section where three combinatorial problems are considered. In general, we can say that the gradient information (reduced costs) can be used to calculate a regret function (see for example [7] for the definition of regret) useful for the variable selection, whereas the optimal assignment in the relaxed problem can be used for the value selection, and finally the lower bound value can be used to select a working subproblem in a local improvement framework, as described in section 5.3.

## 5 Computational Results on Different Problems

In this section we present the empirical results on different problems for which the linear assignment problem turns out to be a relaxation. We report computing times (given in seconds on a Pentium II 200 MHz) and number of fails. We refer to different strategies: (*i*) a pure CP approach exploiting the Branch & Bound *a-la* CP; (*ii*) a strategy exploiting the *LB-based* propagation, referred to as **ST1**; (*iii*) a strategy exploiting both the *LB-based* and *RC-based* propagation, referred to as **ST2**; (*iv*) a strategy exploiting the *LB-based* and *IRC-based* propagation, referred to as **ST3**. Also comparisons with related approaches on the same applications (if any) are shown. The problems considered are: Travelling Salesman Problems instances taken from the TSP-lib and solved also in [6], Timetabling problems described in [7]. Scheduling Problems with setup times are finally considered and solved using a local improvement technique.

Travelling Salesman Problems have been chosen because standard CP techniques perform very poorly on these problems; we are able to solve problems which are one order of magnitude greater than those solved by a pure CP approach. Caseau and Laburthe in [7] have already shown the advantages of CP techniques in Timetabling problems w.r.t. pure OR approaches. Here we show that the tighter integration proposed outperforms their approach. Indeed, the modelling uses different constraints of difference embedding information on cost. These constraints represent different relaxations of the same problem on shared decision variables. Thus, they smoothly interact with each other and with the entire set of problem constraints allowing to efficiently solve the problem. Finally, preliminary results obtained on Scheduling Problems with setup times show the generality of the approach, and propose a new method for modelling and solving such problems. Implementation details, and more computational results on the TSP and Timetabling problems presented can be found respectively in [13] and [11].

### 5.1 TSPs

TSP concerns the task of finding a tour covering a set of nodes, once and only once, with a minimum cost. The problem is strongly NP-hard, and has been

deeply investigated in the literature (see [14] for a survey). Although CP is far from obtaining better results than the ones obtained with state of the art OR technology, it is nevertheless very interesting to build an effective TSP constraint; in fact, many problems contain subproblems that can be described as TSPs, e.g., *Vehicle Routing Problem* (VRP), *Scheduling Problems*, and many variants of TSP are also interesting, e.g., TSP with *Time Windows* (TSPTW). In these cases the flexibility and the domain reduction mechanism of Constraint Programming languages can play an important role, and hybrid CP-OR systems could outperform pure OR approaches (as shown in [16] and [17]).

In this section, a set of symmetric TSP instances (up to 30 nodes, from TSP-lib [20]) is analyzed. The pure CP approach has not been reported because it is not able to prove optimality within 30 minutes on none of the instances considered. Our results have been compared with those achieved by Caseau and Laburthe [6] and reported in row CL97, Table 1. The computing times of this last row are given in seconds on a Sun Sparc 10.

| Problem | gr17 | | gr21 | | gr24 | | fri26 | | bayg29 | | bays29 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Fails | Time | Fails | Time | Fails | Time | Fails | Time | Fails | Time | Fails |
| ST1 | 8.79 | 13k | 0.11 | 96 | 1.7 | 1.5k | 19.88 | 16.6k | 89.4 | 79.8k | 135.7 | 112.8k |
| ST2 | 0.71 | 758 | 0.05 | 31 | 0.28 | 145 | 3.68 | 1.8k | 10.6 | 9.4k | 15.4 | 10.8k |
| ST3 | 0.66 | 646 | 0.06 | 31 | 0.27 | 120 | 2.86 | 1.6k | 11.09 | 7.8k | 13.7 | 8.8k |
| CL97 | 3.10 | 5.8k | 7.00 | 12.5k | 6.90 | 6.6k | 930.0 | 934k | 4.4k | 4.56M | 1.2k | 1.1M |

Table 1: Results on small symmetric TSP instances.

The search strategy used exploits the information coming from the optimization component. It implements a sub-tour elimination branching rule often used in OR-based Branch and Bound algorithm for the TSP. In any stage of the search tree, we consider the solution of the AP, we choose a tour belonging to the optimal AP solution, and we branch by removing one arc of the tour in each branch. Note that the tour chosen, infeasible for the TSP, will not appear in any of the generated branches.

Results show that the use of the back propagation from the objective function to the decision variables (strategies ST2 and ST3) turns out to be very important for efficiently solve optimization problems.

As previously mentioned, one of the interests in solving TSP by Constraint Programming is the flexibility of CP that allow the immediate addition of further constraints to the original problem, e.g., Time Windows, by performing separate propagation on them. In [12] we have shown how to optimally solve TSP with Time Windows by using the path constraint embedding cost-based domain filtering together with well known CP propagation algorithms deriving from the field of scheduling. The resulting algorithm achieves the best known results on some instances, thus being competitive with pure OR approaches.

### 5.2 Timetabling Problems

The timetabling problems considered have been described in [7]. The problems consist in producing a weekly schedule with a set of lessons whose duration goes from 1 to 4 hours. Each week is divided in 4-hours time slots and each

lesson should be assigned to one time slot. The problem involves disjunctive constraints on lessons imposing that two lessons cannot overlap and constraints stating that one lesson cannot spread on two time slots. The objective function to be minimized is the sum of weights taking into account penalties associated to pairs lesson-hour. We have modelled the problem by considering: (*i*) an array of domain variables `Start` representing the course starting times; (*ii*) an array of variables `Slot` representing the slot to which the course is assigned; (*iii*) an array of variables `SingleHours` representing the single hours of each course. Different variables are linked by the following constraints:

```
Start[i] mod 4 = Slot[i]
Start[i] = SingleHours[i][0]
```

Two different matching problems representing two relaxations of the timetabling problem have been modelled by two constraints of difference embedding an optimization component. The first one is the linear assignment relaxation arising when lessons are considered interruptible involving variables `SingleHours`. The cost of assigning each `SingleHours[i]` variable to a value `H` is the cost of assigning the corresponding course to the time slot `H mod 4` divided by the duration of the course. The second relaxation considers variables `Slot` for courses lasting 3 and 4 hours. The corresponding problem is an AP since two 3 or 4 hours courses cannot be assigned to the same slot for limited capacity. The cost of assigning a course to a slot is defined by the problem. The interesting point here is that different problem relaxations coexist and easily interact through shared variables.

In Table 2 we report, in addition to the results of the four described approaches, the results obtained by the constraint *MinWeightAllDifferent* described by Caseau and Laburthe [7]. (In the last row of Table 2, we refer to row 4 of Table 6 of [7], and the corresponding computing times are given in seconds on a Pentium Pro 200 MHz.)

| Problem | Problem 1 | | Problem 2 | | Problem 3 | |
|---|---|---|---|---|---|---|
| | Time | Fails | Time | Fails | Time | Fails |
| Pure-CP | 3.77 | 5.4k | 5.50 | 8.5k | 11.20 | 14.5k |
| ST1 | 0.70 | 213 | 0.15 | 58 | 7.60 | 2.5k |
| ST2 | 0.70 | 199 | 0.10 | 30 | 4.00 | 1.3k |
| ST3 | 0.90 | 182 | 0.16 | 28 | 6.10 | 1.2k |
| CL [7] | 29.00 | 3.5k | 2.60 | 234 | 120.00 | 17k |

Table 2: Results on timetabling instances.

Table 2 shows that for these instances ST2 outperforms in terms of computing times other approaches, although ST3 has more powerful propagation (less number of fails). In this case, in fact, the reduction of the search space does not pay off in terms of computing time.

We have used the information provided by the AP solution also for guiding the search. Defining the regret of a variable as the difference between the cost of the best assignment and the cost of the second best, a good heuristic consists in selecting first variables with high regret. In [7] the regret has been heuristically

evaluated directly on the cost matrix as the difference between the minimum cost and the second minimum of each row (despite of the fact that these two minimum could not be part of the first best and the second best solutions). Reduced cost provide a more accurate computation of the regret: for each variable, a lower bound on the regret is the minimum reduced cost excluding the reduced cost of the value in the AP solution. This regret is then combined in a weighted sum with the size of the domain (following the First-Fail principle), and such a weighted sum is used in the variable selection strategy. Concerning the value selection strategy for variable $X_i$, we have used the solution of the AP.

### 5.3 Scheduling with Set Up Times

We are given a set of $n$ activities $A_1, ..., A_n$ and a set of $m$ unary resources (resources with maximal capacity equal to one) $R_1, ..., R_m$. Each activity $A_i$ has to be processed on a resource $R_j$ for $p_i$ time units. Resource $R_j$ can be chosen within a given subset of the $M$ resources. Activities may be linked together by precedence relations. Sequence dependent setup times exist among activities. Given a setup time matrix $S^k$ (square matrix of dimension equal to $n$), $s^k_{ij}$ represents the setup time between activities $A_i$ and $A_j$ if $A_i$ and $A_j$ are scheduled sequentially on the same resource $R_k$. In such a case, $\text{start}(A_j) \geq \text{end}(A_i) + s^k_{ij}$. Also a setup time $su^k_j$ before the first activity $A_j$ can start on resource $R_k$ may exist. A teardown time $td^k_i$ after the last activity $A_i$ ends on resource $R_k$ may exist.

Constraints of the problem are defined by the resource capacity, the temporal constraints, and the time bounds of the activities (release date, and due date). The goal is to minimize the sum of setup time, given a maximal makespan.

A multiple-TSP *M-TSP* can model a relaxation of the scheduling problem where each resource, and each activity are represented by nodes and arc costs are the setup times. The solution of the M-TSP provides both an assignment of activities to resources and their minimum cost sequencing. Again, the AP can be used to calculate a lower bound on the optimal *M-TSP*, thus to perform pruning on problem variables, and to guide the search.

In the following, we will give some preliminary results. The scheduling problem analyzed were solved in two phases: we first looked for a feasible solution, and then we iteratively select a small time window $TW_i$, we freeze the solution outside $TW_i$, and perform a Branch and Bound search within the selected window. The scheduling problem considered consists in 25 job of 6 activities each. The activities of each job are linked by temporal constraints and the last activity of each job is subject to a deadline. Each activity requires a set of alternative unary resources and a discrete resource with a given capacity profile.

| | Makespan | Total Setup | CPU Time |
|---|---|---|---|
| First Sol. | 2728 | 930 | 8 |
| Pure-CP | 2705 | 750 | 386 |
| ST2 | 2695 | 600 | 249 |

Table 3: Results on a Scheduling Problem with setup times.

The first solution (first row of Table 3) produces a makespan equal to 2728 and a total setup time equal to 930. This first solution is used as starting point for the local improvement phase. The second row of Table 3 reports the improvement on the first solution obtained using a pure CP approach, while the third row reported the results obtained using the optimization constraint (LB-based and RC-based propagation). Both approaches used the same search strategy. The use of the optimization constraint played an important role in the local improvement phase. In fact for a given time window $TW_i$, the lower bound gives very good information on the local optimal solution because the scheduling constraints (relaxed on the *M-TSP*) are locally not tight. Indeed, in some cases the gap between the value of the lower bound calculated at the root node and the value of the local optimal solution found is zero.

In this application the optimization constraint is also very important for the selection of the time window $TW_i$. For each time window $TW_i$ we calculate the gap between the current cost and the lower bound. Such a value is used to select the time window in which running the Branch and Bound optimization. In fact, the higher the gap is, the more chances we have to obtain a good improvement on the solution.

It is important to stress that in this case the optimization constraint interacts with all the scheduling constraints (time bounds, precedence relationship, capacity constraints) thought shared variables. The Edge Finder [1] constraint may, for example, deduce that a given activity $A_i$ must precede a set of other activities, and this information is made available to the optimization constraint.

## 6 Conclusion and Future Work

In this paper, we have proposed the use of an optimization component such as a Linear Program in global constraints. For feasibility purposes, global constraints represent a suitable abstraction of general problems. For optimization purposes embedding OR methods in global constraints is a necessary condition for efficiently handle objective functions.

The advantages of the proposed integration are that we are able to infer primitive constraints starting from non primitive ones on the basis of lower bound and reduced costs information. This enhances operational behaviour of CP for optimization problems by maintaining its flexibility and its modelling capabilities.

Although most of the OR techniques used are fairly standard in the OR community we believe that their introduction in CP global constraints leads to significant new contributions. We greatly powered the CP constraints for optimization problems. We also powered the back-propagation from the objective function to the decision variables; such propagation is limited in a pure OR framework since pure OR branch and bound does not have a constraint store active on shared variables. This last point, in particular, allowed us to easily model and solve problems whose pure OR modelling would lead to very complex algorithms. Finally, the different prospective in which reduced cost fixing is used

brought (and may bring) to new contributions such as the *improved reduced cost* propagation.

Future work concern further generalization of the method by integrating in global constraint a general LP solver providing information on lower bound and on reduced costs. Also, we are currently investigating the use of additive bounds [10] and other specialized cost-based methods in global constraints.

## Acknowledgements

This work has been partially supported by ILOG S.A. (France). The authors are grateful to F. Laburthe, U. Junker, T. Kasper, W. Nuijten, J. Pommier, J.F. Puget, P. Toth and D. Vigo for fruitful discussions and suggestions. Thanks are also due to anonymous referees for useful comments on an earlier version of this paper.

## References

1. P. Baptiste, C. Le Pape, and W. Nuijten. Efficient operations research algorithms in constraint-based scheduling. In *Proceedings of IJCAI'95*, 1995. 201
2. H. Beringer and B. De Backer. Combinatorial problem solving in Constraint Logic Programming with Cooperating Solvers. In C. Beierle and L.Plumer, editors, *Logic Programming: formal Methods and Practical Applications*. North Holland, 1995. 190, 191
3. A. Bockmayr and T. Kasper. Branch-and-Infer: A unifying framework for Integer and Finite Domain Constraint Programming. *INFORMS J. Computing*, 10(3):287 – 300, 1998. 191
4. G. Carpaneto, S. Martello, and P. Toth. Algorithms and codes for the Assignment Problem. In B. Simeone et al., editor, *Fortran Codes for Network Optimization - Annals of Operations Research*, pages 193–223. 1988. 190, 193, 194
5. Y. Caseau and F. Laburthe. Improving Branch and Bound for Jobshop Scheduling with Constraint Propagation. In M. Deza, R. Euler, and Y. Manoussakis, editors, *Combinatorics and Computer Science*, LNCS 1120, pages 129–149. Springer Verlag, 1995. 190, 191
6. Y. Caseau and F. Laburthe. Solving small TSPs with constraints. In *Proceedings of the Fourteenth International Conference on Logic Programming - ICLP'97*, pages 316–330, 1997. 190, 197, 198
7. Y. Caseau and F. Laburthe. Solving various weighted matching problems with constraints. In *Proceedings of CP'97*, 1997. 190, 197, 198, 199
8. G. B. Dantzig. *Linear Programming and Extensions*. Princeton Univ. Press, 1963. 191
9. M. Dell'Amico and S. Martello. Linear assignment. In F. Maffioli M. Dell'Amico and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 355–371. Wiley, 1997. 193
10. M. Fischetti and P. Toth. An additive bounding procedure for the asymmetric travelling salesman problem. *Mathematical Programming*, 53:173–197, 1992. 192, 202

11. F. Focacci, A. Lodi, and M. Milano. Integration of CP and OR methods for Matching Problems. In *CP-AI-OR'99 Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems*, 1999. 197
12. F. Focacci, A. Lodi, and M. Milano. Solving tsp with time windows with constraints. In *ICLP'99 International Conference on Logic Programming*, 1999. 198
13. F. Focacci, A. Lodi, M. Milano, and D. Vigo. Solving TSP through the integration of OR and CP techniques. *Proc. CP98 Workshop on Large Scale Combinatorial Optimisation and Constraints*, 1998. 193, 196, 197
14. M. Jünger, G. Reinelt, and G. Rinaldi. The Travelling Salesman Problem. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*. Wiley, 1997. 198
15. G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988. 190, 192
16. G. Pesant, M. Gendreau, J. Y. Potvin, and J. M. Rousseau. An exact constraint logic programming algorithm for the travelling salesman problem with time windows. *Transportation Science*, 32(1):12–29, 1998. 198
17. G. Pesant, M. Gendreau, J. Y. Potvin, and J. M. Rousseau. On the flexibility of Constraint Programming models: From Single to Multiple Time Windows for the Travelling Salesman Problem. *European Journal of Operational Research*, 117(2):253–263, 1999. 198
18. J. F. Puget. A C++ implementation of CLP. Technical Report 94-01, ILOG Headquarters, 1994. 190
19. J. C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI'94*, 1994. 193
20. G. Reinelt. TSPLIB - a Travelling Salesman Problem Library. *ORSA Journal on Computing*, 3:376–384, 1991. 198
21. R. Rodosek, M. Wallace, and M. T.Hajian. A new approach to integrating Mixed Integer Programming and Constraint Logic Programming. *Annals of Operational Research*, 1997. Recent Advances in Combinatorial Optimization. 190, 191, 192, 194
22. H. Simonis. Calculating lower bounds on a resource scheduling problem. Technical report, Cosytec, 1995. 190, 191