# A Hybrid Exact Algorithm for the TSPTW

Filippo Focacci • Andrea Lodi • Michela Milano

*ILOG SA, 9, Rue de Verdun - 94253 - Gentilly, France*

*D.E.I.S., University of Bologna, Viale Risorgimento, 2 - 40136 - Bologna, Italy*

*D.E.I.S., University of Bologna, Viale Risorgimento, 2 - 40136 - Bologna, Italy*

*ffocacci@ilog.fr* • *alodi@deis.unibo.it* • *mmilano@deis.unibo.it*

---

The *Traveling Salesman Problem with Time Windows* (TSPTW) is the problem of finding a minimum-cost path visiting a set of cities exactly once, where each city must be visited within a specific time window. We propose a hybrid approach for solving the TSPTW that merges Constraint Programming propagation algorithms for the feasibility viewpoint (find a path), and Operations Research techniques for coping with the optimization perspective (find the best path). We show with extensive computational results that the synergy between Operations Research optimization techniques embedded in global constraints, and Constraint Programming constraint solving techniques, makes the resulting framework effective in the TSPTW context also if these results are compared with state-of-the-art algorithms from the literature.

(*Routing; Scheduling; Relaxations; Cutting Planes; Constraint Programming*)

---

## 1. Introduction

The *Traveling Salesman Problem with Time Windows* (TSPTW) is the problem of finding a minimum-cost path visiting a set of cities exactly once, where each city must be visited within a specific time window. The TSPTW has important applications in routing and scheduling, and has been extensively studied (see Desrosiers et al. 1995, for a survey).

In the TSPTW two main components coexist, namely a *Traveling Salesman Problem* (TSP), and a scheduling problem.

In TSPs, *optimization* is usually the most difficult issue: although the feasibility problem of finding a Hamiltonian tour in a graph is NP-hard (Garey and Johnson 1979), in specific applications it is usually "easy" to find feasible solutions, while the optimal one is very "hard" to determine. TSPs have been efficiently solved by using Operations Research (OR) methods such as branch-and-cut (Applegate et al. 1998) and local search (Johnson and McGeoch 1997).

On the other hand, scheduling problems with release dates and due dates usually contain difficult *feasibility* issues since they may involve disjunctive, precedence, and capacity constraints at the same time. The solution of scheduling problems is probably one of the most promising Constraint Programming (CP) area of application to date (see, e.g., Baptiste et al. 1995, Caseau and Laburthe 1994, 1996). This is due to the exploitation of powerful propagation techniques, such as *edge finding* (Carlier and Pinson 1995), in global constraints.

In this paper, we propose a hybrid approach for solving the TSPTW that merges OR techniques for coping with the optimization perspective, and CP propagation algorithms for the feasibility viewpoint. The approach embeds a set of constraint-propagation algorithms that classically derive from feasibility reasoning in scheduling and routing contexts. In addition, we extensively use propagation algorithms exploiting the calculation of bounds to remove from variable domains those values that cannot lead to better solutions than the best one found so far (*cost-based* propagation). This technique, which is standard in OR (the so-called *reduced-cost fixing*), has been introduced by Focacci et al. (1999a) in CP, and has been shown to be particularly effective in this context. Namely, the use of bound calculation for filtering allows: (i) substantially reducing the search space, thus leading to performance improvements with respect to pure CP algorithms; and (ii) better coping with information deriving from the objective function; these aspects are, in general, not effectively handled by CP tools.

Even more importantly, reduced-cost fixing in the CP context turns out to be very well integrated in classical CP methodologies since domain filtering deriving from considerations on costs may trigger other propagation on shared variables, and vice-versa.

We show with extensive computational results that the synergy between OR optimization techniques embedded in global constraints, and CP constraint-solving techniques, makes the resulting framework effective in the TSPTW case. The new algorithm largely outperforms pure CP approaches, can be compared with state-of-the-art OR approaches, and has the big advantage of being easily adapted to variants of the problem, e.g., TSPTW with precedence constraints, with pickup and delivery, with multiple time windows.

The paper is organized as follows. In Section 2 the literature on the TSPTW is briefly reviewed, while in Section 3 preliminaries on Constraint Programming are provided. In Section 4 both OR and CP models for the problem are presented, whereas in Section 5 the set of propagation techniques proposed to solve the problem are discussed. In Section 6 the cost-based propagation is presented, and in Section 7 the use of cutting planes to improve the effectiveness of the cost-based propagation is discussed. Finally, in Section 8 computational results on a large set of instances from the literature (both symmetric and asymmetric) are reported and compared with different approaches.

## 2. Related Literature

The TSPTW is a time-constrained variant of the TSP. It consists of finding the minimum-cost path to be traveled by a vehicle, starting and returning at the same depot, which must visit a set of $n$ cities exactly once. Each pair of cities has an associated travel time. The service at node $i$ should begin within a time window $[a_i, b_i]$ associated with the node. Early arrivals are allowed, in the sense that the vehicle can arrive before the time-window lower bound. However, in this case the vehicle has to wait until the node is ready for the beginning of service. The problem is NP-hard, and Savelsberg (1985) showed that even finding a feasible solution of TSPTW is NP-complete.

This problem can be found in a variety of real-life applications such as routing, scheduling, manufacturing-and-delivery problems, and, for this reason, has been studied both in the OR and CP contexts.

Namely, the first approaches are due to Christofides et al. (1981) and Baker (1983) and consider a variant of the problem where the total schedule time has to be minimized (MPTW). The first paper presents a branch-and-bound algorithm based on a state-space-relaxation approach, whereas the second one is again a branch-and-bound algorithm exploiting a time-constrained critical-path formulation. Langevin et al. (1993) addressed both MPTW and TSPTW by using a two-commodity flow formulation within a branch-and-bound scheme. Dumas et al. (1995) proposed a dynamic-programming approach for the TSPTW that extensively exploits elimination tests to reduce the state space. More recently, a dynamic-programming algorithm has been presented by Mingozzi et al. (1997). The algorithm embeds bounding functions able to reduce the state space (derived by a generalization of the state-space-relaxation technique) and can also be applied to TSPTW problems with precedence constraints. The most recent approaches are due to Balas and Simonetti (2001) and to Ascheuer et al. (2001). Specifically, Balas and Simonetti (2001) proposed a special dynamic-programming approach: under the assumption that, given an initial ordering of the cities, city $i$ precedes city $j$ if $j \geq i + k$ ($k > 0$), the authors discuss and experiment an algorithm that is linear in $n$ and exponential in $k$. For those problems satisfying these conditions, the dynamic-programming procedure finds an optimal solution, while in the other cases it can be used as a linear time heuristic to explore an exponential-size neighborhood exactly.

Ascheuer et al. (2001) considered several formulations for the *asymmetric* version of the problem (among which a new one introduced in the companion paper, Ascheuer et al. 2000), and computationally compared them within a branch-and-cut scheme. The framework incorporates up-to-date techniques tailored for the Asymmetric TSPTW such as data preprocessing, primal heuristics, local search, and variable fixing (obviously, besides the specific separation algorithms).

From the Constraint Programming side, Caseau and Koppstein (1992) have faced a large task-assignment problem where a set of small TSPTW instances are solved as subproblems. More recently, Pesant et al. (1998) solved the TSPTW by enriching a simple CP model with redundant constraints. These additional constraints embed arc-elimination and time-window-reduction algorithms previously proposed in Langevin et al. (1993) and Desrochers et al. (1992), respectively. A variant of the TSPTW, called TSP with *multiple* Time Windows, has been solved by the same authors (Pesant et al. 1999): the CP flexibility is pointed out by showing that this variant can be solved with exactly the same algorithm used for the original problem and slightly adapting the model.

Finally, Focacci, Lodi and Milano used the TSPTW as a case-study in a series of papers (Focacci et al. 1999b, 2000, 2002). Specifically, a small set of TSPTW instances has been solved in Focacci et al. (1999b) so as to show the effectiveness of the cost-based domain filtering approach, in Focacci et al. (2002) for comparing different relaxations within the framework, and in Focacci et al. (2000) for testing a number of possible methods for integrating cutting planes in CP (see Section 7). Starting from these promising computational results (and mainly those in Focacci et al. 1999b) the present paper summarizes the work above by concentrating on the efficient solution of the TSPTW. To this aim, a new hybrid algorithm (whose main ingredients are cost-based domain filtering, the addition of cutting planes, and the extensive use of Lagrangean techniques) is tailored for TSPTW, computationally evaluated on a large set of both symmetric and asymmetric instances and compared with state-of-the-art algorithms from both the OR and CP sides.

# 3. CP Preliminaries

The class of programming languages called *Constraint Programming* combines declarative programming with constraint solving. CP languages make use of *domain variables*, i.e., variables ranging over the domain of possible values that each variable can assume. Variable domains are restricted during the computation according to the constraints, thus achieving the a priori pruning of the search space. Variables are linked by constraints that can be either mathematical constraints (e.g., $X > Y$, $X \neq Y$) or symbolic constraints. Symbolic constraints involve a set of variables and enable a concise modeling of the problem while embedding complex propagation algorithms. For example, a widely used symbolic constraint is *alldifferent*. The application of the constraint *alldifferent* to an array of domain variables $Vars$, ensures that all variables in $Vars$ have a different value. From an operational point of view, such a constraint can be enforced by eliminating value $k$ from the domain of each variable $X_i \in Vars$, as soon as the value $k$ is assigned to $X_j$, $i \neq j$. In addition, more sophisticated propagation can be performed as described in Régin (1994).

The aim of using symbolic constraints is to "isolate" parts that often appear as substructures in applications. This has an obvious interest from a modeling point of view, and, since these parts often have a "clean" structure, such a structure can be exploited for developing specific and effective propagation algorithms.

The propagation algorithm is part of the constraint itself, and is triggered when an event is raised due to a domain modification of one variable involved in the constraint. The event is in general one of the following: removal of a value, reduction of a domain bound, or instantiation of a variable (the domain is reduced to a single value). Thus, as soon as one constraint produces a modification on the domain of one variable, say $X$, all other constraints involving $X$ are awakened, perform propagation on the basis of the current state of the variables' domains, and at the end are suspended to wait for another event.

The constraint-propagation process ends when no more values can be deleted from variable domains. The convergence is guaranteed since domains can only be shrunk and never be enlarged by constraint propagation. Thus, constraint propagation always reaches a fixed point, i.e., a quiescent state of the constraint network. At the end of the constraint propagation process, we have three possible scenarios: (i) a domain becomes empty and a failure occurs; (ii) a solution is found, i.e., all variables are assigned to one value; or (iii) some domains contain more than one value. In this third case, since constraint propagation is not complete, we need a search strategy in order to explore the remaining search tree.

Basically, this step is quite similar to the branching strategies used in OR branch-and-bound algorithms, i.e., the problem is partitioned in subproblems. In the OR context, each of these subproblems is relaxed, the *optimal* solution of the relaxation is computed (bound), and the bound is exploited to prune suboptimal solutions from the solution space. In the CP context, instead, propagation algorithms are applied to each subproblem until the fixed point is reached. Thus, an implicit relaxation is considered but not solved to optimality unless the subproblem itself is solved by either finding a *feasible* solution or proving that none exists. More formally, by restricting consideration, with no loss of generality, to minimization problems, CP systems usually implement a sort of branch-and-bound algorithm in which the main idea is to solve a set of decision (feasibility) problems (i.e., a feasible solution is found if it exists), leading to successively better solutions. In particular, each time a feasible solution $z^*$ is found (whose associated cost is $f(z^*)$), a constraint $f(x) < f(z^*)$, where $x$ is any feasible solution, is added to each subproblem in the remaining search tree. The purpose of the added constraint, called an *upper-bounding constraint*, is to remove portions of the search space that cannot lead to better solutions than the best one found so far. The problem with this approach is twofold: (i) CP does not rely on sophisticated algorithms for computing lower and upper bounds for the objective function, but derives their values starting from the variable domains; (ii) in general, the link between the objective function and the problem decision variables is quite poor and does not produce effective domain filtering.

In this paper we show how to use CP for solving TSPTW instances. CP here is properly extended to maintain its advantages (pruning capabilities, flexibility, and declarative modeling), and overcome its limitations in dealing with optimality reasoning.

# 4. TSPTW Models

In this section we present both OR and CP models for the TSPTW, and in particular, we consider models that refer to the asymmetric version of the problem. Note that the time-window constraints typically lead symmetric (travel) cost matrices to become somehow asymmetric by inducing travel directions.

Recently, Ascheuer et al. (2000) proposed an Integer Linear Programming (ILP) model for the TSPTW based on a graph-theory representation through a directed graph $G = (W, A)$, where the set $W$ contains a node for each city ($n$) plus a node (conventionally, node

0) for the depot. To each node $i \in W \setminus \{0\}$ is associated the time window $[a_i, b_i]$ in which the visit has to be performed. The model involves a binary variable $x_{ij}$ and a cost $t_{ij}$ associated with each arc $(i,j) \in A$, and is as follows:

$$
\begin{aligned}
Z(TSPTW) \quad & = \min \sum_{(i,j)\in A} t_{ij} x_{ij} && (1)\\
\text{subject to:} \quad & x(\delta^+(i)) = 1 \quad \forall\, i \in W && (2)\\
& x(\delta^-(i)) = 1 \quad \forall\, i \in W && (3)\\
& x(A(S)) \leq |S| - 1 \quad \forall\, S \subset W, 2 \leq |S| \leq n && (4)\\
& x(P) \leq |P| - 1 = k - 2 \quad \forall \text{ inf. path } P = (v_1, \ldots, v_k) && (5)\\
& x_{ij} \in \{0,1\} \quad \forall\, (i,j) \in A && (6)
\end{aligned}
$$

where for each subset $S \subset W$, $A(S)$ represents the set of arcs connecting nodes in $S$, while $\delta^+(i)$ (resp. $\delta^-(i)$) represents the set of arcs whose starting (resp. ending) node is $i$. Let $B \subseteq A$ be any set of arcs. Then, $x(B)$ denotes the sum of all $x_{ij}$ such that $(i,j) \in B$. Finally, $x(P)$ denotes the sum of the variables corresponding to a path $P$, i.e., $x(P) = \sum_{i=1}^{k-1} x_{v_i, v_{i+1}}$, where $v_i \in \{0, 1, \ldots, n\}$.

Constraints (2)-(3) are degree constraints assuring that each node (city) be visited exactly once, whereas constraints (4) are the so-called *subtour elimination constraints* (SECs), and guarantee connectivity. Finally, constraints (5) are called *infeasible-path-elimination constraints*, and forbid paths in which the visit in a given node $i$ violates the time window $[a_i, b_i]$.

A CP model of the TSPTW has been previously proposed in Pesant et al. (1998); in this section, we recall it, and the set of constraints used to enforce feasibility. Let $V = \{0, 1, 2, \ldots, n, n+1\}$ be a set of nodes, where nodes $1, \ldots, n$ represent the cities to be visited, and nodes 0 and $n+1$ represent the origin and destination depots. Each node $i \in V \setminus \{0, n+1\}$ has an associated time window $[a_i, b_i]$ representing the time frame during which the service at city $i$ must be executed. Moreover, each node has an associated duration $dur_i$ representing the duration of the service at city $i$. For each pair of cities $(i,j)$, $t_{i,j}$ indicates the travel cost from $i$ to $j$.

We define four domain variables per node. Domain variables $Next_i$ and $Prev_i$ identify cities visited after and before node $i$, respectively. The domain of variables $Next_i$ contains values $[1..n+1]$, except for variable $Next_{n+1}$ which is set to 0, while the domain of each $Prev_i$ contains values $[0..n]$, except for variable $Prev_0$, which is set to $n+1$. Domain variable

$Cost_i$ identifies the cost to be paid to go from node $i$ to node $Next_i$, i.e., $t_{i,Next_i}$. Finally, domain variable $Start_i$ identifies the time at which the service begins at node $i$. With no loss of generality, we can impose $Start_0 = 0$, and $Cost_{n+1} = 0$.

A feasible solution of the TSPTW is an assignment of a different value to each variable that avoids tours and respects time-window constraints. An optimal solution of the problem is the one minimizing the sum of the travel costs $t_{i,Next_i}$.

Using the introduced variables, a CP model for TSPTW is the following:

$$Z(TSPTW) = \min \sum_{i \in V} Cost_i \tag{7}$$

$$\text{subject to:} \quad alldifferent([Next_0, ..., Next_{n+1}]) \tag{8}$$

$$nocycle([Next_0, ..., Next_n]) \tag{9}$$

$$Next_i = j \Leftrightarrow Prev_j = i \ \forall i, j \in V \tag{10}$$

$$Next_i = j \Rightarrow Cost_i = t_{i,j} \ \forall i, j \in V \tag{11}$$

$$Next_i = j \Rightarrow Start_i + dur_i + Cost_i \leq Start_j \ \forall i, j \in V \tag{12}$$

$$a_i \leq Start_i \leq b_i \ \forall i \in V \setminus \{0, n+1\} \tag{13}$$

The *alldifferent* constraint (8) imposes that each node has exactly one outgoing and exactly one incoming arc, i.e., each city is visited exactly once. The constraint *nocycle* (9) forbids tours. (Specifically, since constraint (9) does not involve node $n+1$, all cycles, and not only subcycles, are excluded, and it does not subsume constraint (8).) Constraints (10) create the link between the *Next* and *Prev* variables; constraints (11) create a link between the travel costs and the *Cost* variables. More interesting are constraints (12) that define a connection between the TSP model (*Next* and *Prev* variables) and the scheduling model (*Start* variables) with sequence dependent set-up times ($t_{i,j}$). Finally, constraints (13) simply define the bounds on the *Start* variables by imposing that the service in a node must start within the node time window.

Note that the model (7)-(13) involves *redundant* constraints, i.e., constraints that are not necessary to guarantee feasibility. The use of redundant constraints is very effective in the solution phase, since they interact through shared variables, and, by capturing different aspects of the overall problem, produce a coordinated propagation.

**CP Model Implementation.** The above model has been implemented by creating a mapping between the TSPTW and a scheduling problem with a single unary resource and sequence-dependent set-up times. In the mapping, each visit (city) corresponds to an activity. The start-time variable of activity $i$ corresponds to $Start_i$ in the TSPTW, and its duration is equal to $dur_i$. A transition-time matrix is defined based on matrix $t_{i,j}$, which enforces that the beginning of activity $j$ is after the end of activity $i$ plus $t_{i,j}$ if activity $j$ is scheduled immediately after activity $i$.

The flexibility of Constraint Programming allows a profitable interaction between the routing and scheduling propagation techniques (see Section 5).

# 5. Propagation Techniques

In this section we define the propagation scheme used for solving the TSPTW. We have exploited both well-known CP filtering algorithms and problem-oriented propagation techniques.

Concerning the *alldifferent* constraint, we used the well-known filtering algorithm proposed by Régin (1994), and based on cardinality reasoning over subsets.

For the *nocycle* constraint, we have implemented the simple, but effective, propagation described in Caseau and Laburthe (1997) and Pesant et al. (1998). Specifically, as soon as a variable $Next_i$ (resp. $Prev_j$) is instantiated to value $j$ (resp. $i$), we remove from the variable representing the end of the partial path starting in $j$ the value representing the start of the partial path ending in $i$.

The set of equations (10) are propagated by considering the counterpart of the constraint: whenever a value $j$ is removed from the domain of variable $Next_i$, the value $i$ is removed from the domain of variable $Prev_j$, and vice-versa.

The set of constraints (11) is maintained by forcing

$$\min_{k \in Dom(Next_i)} t_{i,k} \leq Cost_i \leq \max_{k \in Dom(Next_i)} t_{i,k} \quad \forall\, i \in V$$

where $Dom(V)$ identifies the domain of variable $V$.

Analogous propagation is performed for constraints (12):

$$\min_{k \in Dom(Prev_i)} \{Start_{Min_k} + dur_k + t_{k,i}\} \leq Start_i \quad \text{and}$$
$$Start_i \leq \max_{k \in Dom(Next_i)} \{Start_{Max_k} - t_{i,k} - dur_i\} \quad \forall\, i \in V$$

where $Start_{Min_k}$ and $Start_{Max_k}$ identify the lower and upper bounds of variable $Start_k$, respectively.

Additional propagation on the $Next$ variables is performed by testing if

$$Start_{Min_i} + dur_i + t_{i,j} > Start_{Max_j} \quad \forall i, j \in V$$

Whenever this test is successful for some pair $(i, j)$, the value $j$ can be removed from the domain of variable $Next_i$.

A more powerful propagation can be performed by maintaining the set of all nodes $\mathcal{B}_i$ (resp. $\mathcal{A}_i$), which must precede (resp. follow) node $i$ (see Langevin et al. 1993 for details). If, for a given pair of nodes $(i, j)$, the set of successors of $i$ and the set of predecessors of $j$ have non-empty intersection ($\mathcal{A}_i \cap \mathcal{B}_j \neq \emptyset$) this implies that at least a third node $k$ must be inserted between nodes $i$ and $j$, and therefore we can remove value $j$ from the domain of variable $Next_i$, and value $j$ from the one of variable $Prev_j$.

The sets $\mathcal{B}_i$ and $\mathcal{A}_i$ can be calculated as:

$$\mathcal{B}_i = \{k \in V | Start_{Min_i} + sp_{i,k} > Start_{Max_k}\}$$
$$\mathcal{A}_i = \{k \in V | Start_{Min_k} + sp_{k,i} > Start_{Max_i}\}$$

where $sp_{i,j}$ is the shortest path in terms of travel cost from $i$ to $j$. The computation of the shortest path from any node to any other has a complexity of $O(n^3)$. We have experimentally tested this propagation algorithm by efficiently re-computing, at each node of the search tree, the shortest paths. Since the trade-off between the propagation obtained and the increasing of computing times was not satisfactory, we decided to avoid this calculation. However, since our overall approach is implemented using ILOG Solver and Scheduler (ILOG 2000a, 2000b) (see Section 8), we used for the sets $\mathcal{B}_i$ and $\mathcal{A}_i$ the ones defined by the *Precedence Graph Constraint* of ILOG Scheduler which maintains, for each activity, its relative position with respect to the other activities on a given resource. For example, for an activity $i$ we can ask for the set of activities that surely precede (resp. follow) $i$ in the resource, or activities that are still unranked with respect to $i$ (i.e., may precede or follow). These sets are gathered by joining the information associated with time bounds, temporal constraints, disjunctive constraints, and temporal closure, and are used to build the sets $\mathcal{A}$ and $\mathcal{B}$.

Additional propagation on variables based on sets $\mathcal{B}_i$ and $\mathcal{A}_i$ and shortest paths is proposed in Desrochers et al. (1992), but, after preliminary computational tests and due to its limited effectiveness, we decided to not use it.

## 6. Cost-Based Domain Filtering

In this section, we describe the cost-based domain-filtering technique introduced in Focacci et al. (1999a), where it is computationally tested on small TSPs, matching and scheduling problems with set-up times.

The idea is to create a global constraint embedding a propagation algorithm aimed at removing from variable domains those assignments that do not improve the best solution found so far. Domain filtering is achieved by optimally solving a problem that is a relaxation of the original problem. In this paper, we consider the *Assignment Problem* (AP) (see Dell'Amico and Martello (1997) for a recent annotated bibliography) as a relaxation of the TSP (and, consequently, of the TSPTW) since AP has been shown to be particularly effective in this context. (In Focacci et al. (2002) we have tested the cost-based domain filtering idea by using also other relaxations, e.g., the Minimum Spanning Arborescence.)

The AP is the graph-theory problem of finding a set of *disjoint* subtours in a directed graph such that all the nodes are visited exactly once, and the overall cost is at a minimum. An ILP model for AP is the one defined by constraints (1)-(3) and (6), and an optimal *integer* solution of AP can be computed in $O(n^3)$ time by using the *primal-dual* Hungarian algorithm (see Carpaneto et al. 1988 for details). It is easy to see that if this integer solution is composed by exactly one tour, then it is optimal for the TSP.

The AP relaxation provides: (i) the optimal AP solution, i.e., a variable assignment; (ii) the value of this solution which is a lower bound $LB$ on the original problem; and (iii) a reduced-cost matrix $\bar{t}$. Each $\bar{t}_{ij}$ estimates the additional cost to be added to $LB$ if variable $Next_i$ is assigned to $j$. The lower bound value $LB$ is trivially linked to the variable representing the objective function $Z$ through the constraint $LB \leq Z$. More interesting is the propagation based on reduced costs. Given the reduced-cost matrix $\bar{t}$ of element $\bar{t}_{ij}$, it is known that $LB_{ij} = LB + \bar{t}_{ij}$ is a valid lower bound for the problem in which variable $Next_i$ is forced to the value $j$. Therefore, the following propagation holds

$$LB_{ij} > Z_{max} \;\Rightarrow\; Next_i \neq j \;\; \forall\, i, j \in V$$

where $Z_{max}$ is the current upper bound on the solution value.

The events triggering this propagation are changes in the upper bound of the objective function variable $Z$, and any change in the problem variable domains that involves an assignment belonging to the solution of the current relaxation. As mentioned, the solution of the first AP relaxation has a complexity of $O(n^3)$, whereas each following AP re-computation due to a domain reduction can be efficiently computed in $O(n^2)$ time through a single augmenting path step (see Carpaneto et al. 1988 for details). The reduced-cost matrix is obtained without extra computational effort during the AP solution, thus the time complexity of the filtering algorithm is $O(n^2)$.

Since $LB_{ij}$ is just a lower bound on the new AP solution corresponding to forcing the assignment $Next_i = j$, it is easy to see that improving it could trigger additional propagation. In particular, some bounds on the cost of the augmenting path connecting the previous nodes $Next_i$ and $Prev_j$ have been proposed by Focacci et al. (2002), and are used here to propagate further. Since these bounds, based on storing row and column minima, can again be computed in $O(n^2)$ time, they do not increase the time complexity of the overall filtering algorithm.

Reduced-cost fixing is a classical component of OR methods, but it seems to assume a different flavor in CP, appearing to be particularly suited to this context. Sophisticated OR exact methods are based on the solution of relaxations of the original problem. In this context reduced-cost fixing is a by-product of the solution technique. In a sense, reduced-cost fixing is *subsumed* by the optimal solution of the relaxation. On the contrary, reduced costs in CP represent additional information that is not derived by traditional CP problem solving activity. In addition, in CP reduced-cost fixing triggers other constraint propagation, thus further reducing the search space.

# 7. Improving Cost-Based Propagation Through Cutting Planes

It is easy to see that the cost-based technique could benefit from an improvement of the quality of the used bound. Tighter bounds obviously produce a more powerful reduction of the solution space, but the overall effects are smaller computing times only if the computation of these bounds can be efficiently performed.

A classical OR method for obtaining good bounds is to use the LP relaxation in conjunction with *cutting planes* (or simply "cuts") so as to tighten the formulation. Focacci et al. (2000) experimented with several ways of using cutting planes within a CP framework, namely the addition of cutting planes either just at the root node of the decision tree, or at each node, and a hybrid approach in which the cuts are added at the root node and relaxed in Lagrangean fashion during the search. The idea of the third strategy, which is discussed in detail in the following sections, is quite simple: we want to obtain an improved bound through the addition of cutting planes, but we want to solve, during the search, an AP relaxation since it can be "fast" and incrementally computed.

In the next two sections we describe how the AP relaxation can be tightened by the addition of cuts, and the method based on the Lagrangean relaxation of these cuts.

## 7.1 Improving the AP Relaxation

As pointed out in Section 6, the AP relaxation is obtained from model (1)-(6) by relaxing (removing) SECs and infeasible path constraints. In this way we obtain an integer linear program with some *structure* (the AP), which can be solved to optimality through either a special purpose algorithm (e.g., the Hungarian algorithm) or a general LP solver since the integrality requirements are, in this case, unessential. Since, in general, the optimal solution of the AP is not feasible for the TSPTW, in order to improve this bound we need to solve the so-called *separation problem* iteratively. More formally, given the optimal solution of a linear relaxation of TSPTW, say $x^*$, we call *separation* the problem of finding a linear inequality $\alpha^T x \leq \alpha_0$ valid for each feasible solution of TSPTW, and such that $\alpha^T x^* > \alpha_0$, or saying that none exists. Since the TSPTW is NP-hard, the separation problem as defined above is also NP-hard (see, Grötschel et al. 1988). However, the separation can be polynomially solvable for some special *class* of inequalities (a family of valid constraints sharing a special structure), and in the other cases some heuristics can be used.

In this paper, the AP bound is improved by the addition of: (i) SECs and (ii) *Sequential Ordering Problem* inequalities (SOPs).

**SEC Inequalities.** The separation problem for SECs can be solved in polynomial time by computing the *minimum capacity cut* in the graph induced by $x^*$ and the same holds for each following LP obtained by the iterative addition of cuts. For the separation problem we use an implementation (Jünger et al. 2000) of the *mincut* algorithm by Padberg and Rinaldi (1990) to compute a set of subtour-elimination cuts. Note that, although the Padberg and Rinaldi procedure is specific for undirected graphs, i.e., symmetric TSPs, it can also be used for asymmetric TSPs via a simple transformation (see Fischetti and Toth 1997).

**SOP Inequalities.** The precedence-constrained TSP, also known as the *Sequential Ordering Problem*, is a relaxation of the TSPTW, so each valid inequality for it can be used for the solution of the TSPTW. In particular, we consider the following valid inequalities studied by Balas et al. (1995) in the SOP context. With the notation $x(H, K)$ ($H, K \subset W \setminus \{0\}$), we indicate the sum of the $x$ variables of all arcs in $A$ with starting node in $H$ and ending node in $K$.

(i) *Predecessor inequalities* (14) ($\pi$ inequalities). Let $S \subseteq W \setminus \{0\}$, $\bar{S} := W \setminus \{0\} \setminus S$, then

$$x((S \setminus \pi(S)), (\bar{S} \setminus \pi(S))) \geq 1 \tag{14}$$

where $\pi(S)$ indicates the set of nodes which must *precede* the nodes in $S$.

(ii) *Successor inequalities* (15) ($\sigma$ inequalities). Let $S \subseteq W \setminus \{0\}$, $\bar{S} := V \setminus \{0\} \setminus S$, then

$$x((\bar{S} \setminus \sigma(S)), (S \setminus \sigma(S))) \geq 1 \tag{15}$$

where $\sigma(S)$ indicates the set of nodes which must *follow* the nodes in $S$.

For the separation of both $\pi$ and $\sigma$ inequalities, we use the heuristic procedure proposed in Balas et al. (1995) and the corresponding computer code by Ascheuer et al. (2001).

## 7.2 Lagrangean Relaxation of Cuts

As mentioned at the beginning of Section 7, there are several ways of exploiting the cuts described in the previous section (and, possibly, some additional classes). However, the explicit use of cuts during the extensive enumeration performed by a CP framework could be less effective than in the OR context. Focacci et al. (1999a) have shown that the cost-based domain-filtering approach is suited to CP when used in conjunction with a special-purpose algorithm able to solve the corresponding relaxation fast and incrementally. In the TSPTW case, as soon as a cut is added to the AP relaxation in order to improve it, the drawback is that the obtained linear program is no longer an AP, thus from that point the relaxation has to be solved through a general-purpose LP solver.

An alternative way of exploiting cuts is to add them just at the root node of the branch decision tree, and then relaxing them in Lagrangean fashion in order to obtain again an AP as a relaxation. (See, e.g., Sellmann and Fahle 2001 and Benoist et al. 2001 for other references of Lagrangean techniques in CP.) Let $L_R$ denote the final linear problem obtained by adding to the AP model (1)-(3) (plus variable bounds) the set of separated constraints, and let $LB_R$ be the final lower bound value obtained by solving $L_R$ to optimality. If $L_R$ has been solved through the simplex method, the optimal value of the *dual* variables associated with every constraint is returned by the LP solver. Moreover, for the LP duality theory, the vector of dual values returned by the LP solver coincides with the optimal vector of *Lagrangean multipliers*, i.e., with the vector of Lagrangean multipliers producing the best (highest) lower bound.

Formally, the framework is as follows. Let $C$ denote the set of constraints in the form $\alpha^T x \leq \alpha_0$ added to the AP to obtain $L_R$. If for each constraint $\alpha_i^T x \leq \alpha_{i0}$ ($\forall i \in C$), we relax it in Lagrangean fashion:
(i) by multiplying it by the Lagrangean multiplier $\lambda_i$ ($\geq 0$), and
(ii) by summing the result to the objective function,
we obtain a new AP (denoted with $\text{AP}_R$) having objective function

$$\min \sum_{(i,j)\in A} t_{ij}x_{ij} + \sum_{i\in C} \lambda_i(\alpha_i^T x - \alpha_{i0}) = -\sum_{i\in C} \lambda_i \alpha_{i0} + \min \sum_{(i,j)\in A} \widetilde{t}_{ij}x_{ij}$$

i.e., an AP with modified cost matrix $\widetilde{t}$. Finally, if the value of each $\lambda_i$ ($\forall i \in C$) is set equal to the value of the dual variable associated with the constraint $i$ (i.e., we use the *optimal* Lagrangean multipliers), the solution value obtained by solving $AP_R$ through the Hungarian algorithm coincides with $LB_R$, and the $x^*$ vector is integer. (Since the cuts are in $\leq$ form, the corresponding dual variables are non-positive, thus they must be inverted to be used as Lagrangean multipliers.)

The integer solution of $AP_R$ computed through the Hungarian algorithm can be seen as an advanced (more accurate) starting point for the cost-based domain-filtering algorithm: at each node of the search tree the relaxation provided by $AP_R$ is considered and modified according to branch decisions and variable fixing, but no more cuts nor Lagrangean multipliers are re-computed. The intuition for adding the cuts just at the root node and relaxing them in Lagrangean fashion is the following. We are confident, within a CP framework, to be able to explore efficiently even a huge number of branching nodes, thus we exploit cost-based propagation through a special-purpose, fast and incremental algorithm (the Hungarian method for AP). However, we want to reduce *preliminarily* as much as possible the search tree by using cut generation at the root node. The relaxation in Lagrangean fashion of the cuts can have, however, some drawbacks during the search. The Lagrangean multipliers associated to the cuts are fixed to values that are optimal at the root node, but could be "far" from the optimal ones during the search, and no re-optimization (*subgradient optimization*) is performed. In particular, if during the search, a cut $\alpha^T x \leq \alpha_0$, which was tight at the root node (i.e., $\alpha^T x = \alpha_0$), become trivially satisfied by the current partial instantiation, i.e., $\alpha^T x < \alpha_0$, its contribution to the objective function is a *positive* value, hence a *penalty* with respect to the same solution where the cut is removed. Thus, while at the root node, the bound produced by the Lagrangean relaxation is in general much better (higher) than the bound obtained by solving the AP relaxation (without cuts), during the search if cuts become no longer tight, the bound decreases and, at some point, it may becomes worse than the pure AP bound.

In any case, the bound obtained by solving $\text{AP}_R$ is always a valid lower bound for the problem, and the above drawback can be reduced by performing some kind of *purging* in order to disable during the search those cuts that are no longer necessary, i.e., both not tight and trivially satisfied.

Suppose, for example, that the following SEC has been added at the root node to the problem formulation:

$$x_{12} + x_{21} + x_{13} + x_{31} + x_{23} + x_{32} \leq 2 \tag{16}$$

so as to avoid the subtour involving nodes 1, 2 and 3. The following two examples show cases in which we are allowed to remove the Lagrangean relaxation of (16) from $\text{AP}_R$:

(i) during the search the CP variable $Next_1$ is instantiated to value 2 (thus, $x_{12} = 1$ and $x_{13} = 0$), values 1 and 3 are removed from the domain of variable $Next_2$ (thus, $x_{21} = 0$ and $x_{23} = 0$), and values 1 and 2 are removed from the domain of variable $Next_3$ (thus, $x_{31} = 0$ and $x_{32} = 0$). Then, in the remaining subtree constraint (16) is trivially satisfied, and no interesting information is associated to it;

(ii) during the search variable $Next_1$ is instantiated to value 5. Since constraint (16) is a SEC, we know that no subtour in a subset of nodes $S \subset W$ can exist if there is an arc crossing the subset.

In the first example, the cut purging is based on a pure *algebraic* reasoning, while in the second example the *knowledge of the structure* of the cut (and of the problem) is exploited. (Note that the concept of purging is standard for OR branch-and-cut, cuts that are no longer "effective" are removed and included in the *cutpool* so as to keep the LP small, but in case of Lagrangean relaxation of cuts this operation is even more important since it affects the value of the bound.)

Our cost-based algorithm adopts both strategies to purge cuts that are no longer necessary, and in principle a purging operation should be performed each time a variable $x_{ij}$ is instantiated. However, the purging operation can lead to a relevant update of the cost matrix, thus requiring the computation from scratch of the AP solution ($O(n^3)$ time instead of the simple $O(n^2)$). We adopted specific policies to drive the frequency of the purging, so as to obtain a good computational performance.

# 8. Computational Results

We have implemented and tested the overall approach by using ILOG Solver and Scheduler (ILOG 2000a, 2000b). The algorithm runs on a PC Pentium III 700 MHz (128 MByte of RAM) using ILOG Cplex 6.5 as LP solver, and has been experimented on two different sets of instances from the literature.

**Branching Strategies.** We implemented simple branching strategies that are typical of the scheduling context. In this field, very often a solution is built in chronological order: at each node of the search tree the activity with the minimum *earliest start* time is chosen and scheduled. Hence, all nodes are sequenced by starting with the assignment of a node to the initial depot, until the last node is assigned to the final depot. This strategy (*forward* fashion) has an obvious counterpart: the activity with the minimum *latest end* time is initially assigned to the final depot, and the sequence is built in *backward* fashion. We heuristically choose, by a simple preprocessing, which scheduling strategy to adopt. In both cases, we choose the following (resp. previous) node by exploiting the information provided by the reduced costs: among all values having zero reduced cost (there always exists at least one), we select the one corresponding to the node with the minimum latest start (resp. earliest end) time.

**Asymmetric Instances.** The asymmetric TSPTW instances considered are the `rbg` real-world instances introduced by Ascheuer (1995), and deriving "from an industry project with the aim to minimize the unloaded travel time of a stacker crane within an automated storage system". We consider a set of 40 instances with up to 125 nodes, and the results are reported in Table 1.

For each instance, Table 1 reports its name (`name`), and the number of nodes ($n$). For

Table 1: TSPTW **Asymmetric** Instances (Computing Times in Seconds).

| | | Ascheuer et al. 2001 | | | AP-bound | | | | Lagrangean-bound | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| `name` | $n$ | rLB | Val | Time | rLB | UB | Time | Fails | rLB | UB | Time | Fails |
| rbg010a | 12 | 148 | 149 | 0.1 | 148 | 149 * | 0.0 | 6 | 148 | 149 * | 0.1 | 11 |
| rbg016a | 18 | 177 | 179 | 0.2 | 167 | 179 * | 0.1 | 21 | 167 | 179 * | 0.1 | 21 |
| rbg016b | 18 | 133 | 142 | 8.8 | 126 | 142 * | 0.1 | 27 | 129 | 142 * | 0.2 | 106 |
| rbg017.2 | 17 | 107 | 107 | 0.0 | 100 | 107 * | 0.0 | 17 | 107 | 107 * | 0.1 | 28 |
| rbg017 | 17 | 148 | 148 | 0.8 | 124 | 148 * | 0.1 | 30 | 138 | 148 * | 0.1 | 27 |
| rbg017a | 19 | 146 | 146 | 0.1 | 143 | 146 * | 0.1 | 22 | 146 | 146 * | 0.1 | 25 |
| rbg019a | 21 | 217 | 217 | 0.0 | 217 | 217 * | 0.0 | 14 | 217 | 217 * | 0.1 | 14 |
| rbg019b | 21 | 180 | 182 | 54.6 | 175 | 182 * | 0.2 | 80 | 175 | 182 * | 0.2 | 184 |
| rbg019c | 21 | 182 | 190 | 8.7 | 179 | 190 * | 0.3 | 81 | 180 | 190 * | 0.7 | 189 |
| rbg019d | 21 | 343 | 344 | 0.7 | 338 | 344 * | 0.0 | 32 | 338 | 344 * | 0.1 | 538 |
| rbg020a | 22 | 210 | 210 | 0.2 | 207 | 210 * | 0.0 | 9 | 209 | 210 * | 0.1 | 10 |
| rbg021.2 | 21 | 182 | 182 | 0.2 | 179 | 182 * | 0.2 | 44 | 180 | 182 * | 0.3 | 52 |
| rbg021.3 | 21 | 178 | 182 | 27.1 | 166 | 182 * | 0.4 | 107 | 168 | 182 * | 0.5 | 198 |
| rbg021.4 | 21 | 177 | 179 | 5.8 | 166 | 179 * | 0.3 | 124 | 167 | 179 * | 0.4 | 121 |
| rbg021.5 | 21 | 167 | 169 | 6.6 | 160 | 169 * | 0.2 | 55 | 161 | 169 * | 0.3 | 65 |
| rbg021.6 | 21 | 133 | 134 | 1.3 | 126 | 134 * | 0.9 | 318 | 128 | 134 * | 0.7 | 2k |
| rbg021.7 | 21 | 128 | 133 | 4.3 | 125 | 133 * | 0.6 | 237 | 128 | 133 * | 1.0 | 2k |
| rbg021.8 | 21 | 129 | 132 | 17.4 | 125 | 132 * | 0.8 | 222 | 128 | 132 * | 0.6 | 297 |
| rbg021.9 | 21 | 128 | 132 | 26.1 | 125 | 132 * | 0.8 | 310 | 128 | 132 * | 0.9 | 536 |
| rbg021 | 21 | 182 | 190 | 8.5 | 179 | 190 * | 0.3 | 81 | 180 | 190 * | 0.7 | 189 |
| rbg027a | 29 | 266 | 268 | 2.2 | 260 | 268 * | 0.2 | 53 | 264 | 268 * | 0.3 | 50 |
| rbg031a | 33 | 328 | 328 | 1.7 | 275 | 328 * | 12.9 | 4k | 319 | 328 * | 2.7 | 841 |
| rbg033a | 35 | 433 | 433 | 1.8 | 402 | 433 * | 1.0 | 480 | 418 | 433 | – | – |
| rbg034a | 36 | 401 | 403 | 1.0 | 391 | 403 * | 55.2 | 13k | 399 | 403 * | 777.1 | 776k |
| rbg035a.2 | 37 | 158 | 166 | 1.8 | 155 | 166 * | 36.8 | 5k | 155 | 166 * | 38.4 | 5k |
| rbg035a | 37 | 254 | 254 | 64.8 | 221 | 254 * | 3.5 | 841 | 248 | 254 * | 11.9 | 3k |
| rbg038a | 40 | 466 | 466 | 4232.2 | 466 | 466 * | 0.2 | 49 | 466 | 466 * | 0.6 | 998 |
| rbg040a | 42 | 355 | 386 | 751.8 | 282 | 386 * | 738.1 | 136k | 316 | 386 | – | – |
| rbg041a | 43 | 361 | [382,417] | – | 316 | 403 | – | – | 352 | 404 | – | – |
| rbg042a | 44 | 394 | [409,435] | – | 370 | 411 * | 149.8 | 19k | 382 | 411 * | 1036.4 | 177k |
| rbg048a | 50 | 454 | [455,527] | – | 442 | 492 | – | – | 444 | 500 | – | – |
| rbg049a | 51 | 408 | [418,501] | – | 403 | 488 | – | – | 403 | 492 | – | – |
| rbg050a | 52 | 414 | 414 | 18.6 | 405 | 414 * | 180.4 | 19k | 409 | 414 * | 95.6 | 38k |
| rbg050b | 52 | 447 | [453,542] | – | 435 | 551 | – | – | 446 | 527 | – | – |
| rbg050c | 52 | 507 | [509,536] | – | 494 | 543 | – | – | 504 | 544 | – | – |
| rbg055a | 57 | 813 | 814 | 6.4 | 799 | 814 * | 2.5 | 384 | 809 | 814 | – | – |
| rbg067a | 69 | 1047 | 1048 | 5.9 | 1033 | 1048 * | 4.0 | 493 | 1043 | 1051 | – | – |
| rbg086a | 88 | 1042 | [1049,1052] | – | 994 | 1094 | – | – | 1023 | 1086 | – | – |
| rbg092a | 94 | 1084 | [1102,1111] | – | 1061 | 1150 | – | – | 1080 | 1109 | – | – |
| rbg125a | 127 | 1402 | 1410 | 229.8 | 1335 | 1453 | – | – | 1383 | 1446 | – | – |

the branch-and-cut algorithm of Ascheuer et al. (2001) we report the lower bound value at the root node (rLB), the optimal solution value (or the best final [lower,upper] bounds) (Val) and a column (Time) indicating either the computing time on a Sun Sparc Station 10 if optimality is reached or a "–" otherwise. Moreover, the results of our algorithm are reported for the case in which the simple *AP bound* is used in the cost-based algorithm, and for the one in which the *Lagrangean bound* is used. In both cases, Table 1 reports the root lower bound value (rLB), the value of the best solution found (with a "*" if this solution is proven to be optimal) (UB), a column (Time) indicating either the computing time on a PC Pentium III 700 MHz if optimality is reached or a "–" otherwise, and the number of fails (Fails). A time limit of 1,800 CPU seconds has been given to both versions of our algorithm, and when this limit is reached the corresponding solution value is possibly not optimal (and the number of fails is not reported).

The behavior of our algorithms is quite satisfactory. From one side, all the instances that were solved to optimality by the branch-and-cut, but `rbg125a`, can also be solved by the AP-bound version of our code. Instead, this version our algorithm optimally solves instance `rbg042a` and improves the best known solution for instances `rbg041a`, `rbg048a` and `rbg049a`. From the other side, the Lagrangean-bound version fails to solve optimally within the time limit instances `rbg033a`, `rbg055a` and `rbg067a`, solves instance `rbg042a`, and improves the best solution found by the branch-and-cut for instances `rbg041a`, `rbg048a`, `rbg049a`, `rbg50b` and `rbg92a`. Since the branch-and-cut algorithm runs on a Sun Sparc Station 10, a fair comparison of the computing times is not easy. However, the time limit given for the computation in Ascheuer et al. (2001) was 5 hours, so we feel that 1,800 seconds is a reasonable computing time.

Table 1 shows that, for this set of instances, the Lagrangean-bound version is slightly less effective than the AP-bound one. Although the improvement of the initial lower bound is relevant (the root lower bound is not "far" from the one of the branch-and-cut algorithm), the algorithm does not seem to benefit from this advanced starting point while it suffers from the increase of the computational complexity. Indeed, the AP bound is already reasonably good and, used within the cost-based filtering framework, it turns out to be enough on the medium-size problems for which our current implementation is tailored. (Ten large-size instances with up to 233 nodes are also described in Ascheuer (1995).)

Finally, the improvements obtained for some of the unsolved instances are significant mainly because our current implementation does not use a primal heuristic (while an effective one is used in Ascheuer et al. 2001). This suggests that our framework, possibly enhanced by the addition of a primal heuristic, could be used for finding good approximate solutions.

**Symmetric Instances.** The symmetric TSPTW instances considered are the `rc` instances proposed by Solomon (1987) in the Vehicle Routing Problem with Time Windows (VRPTW) context. In particular, the TSPTW instances are obtained by considering the single-vehicle decomposition deriving from VRPTW solutions in the literature (i.e., Rochat and Taillard 1995 and Taillard et al. 1995). This decomposition generates instances with up to 44 cities that have been recently considered by Pesant et al. (1998). Since for these instances the travel times among the cities are computed as Euclidean distances, a relevant point concerns the precision. Pesant et al. (1998) have shown that an optimal solution computed with distances truncated to two decimal places may even become infeasible by allowing a precision of *four* decimal places. (In the same paper, it is also pointed out that CP-based algorithms do not suffer from an increase of precision, while dynamic programming approaches obviously do.) For this reason, and to allow a fair comparison, we also considered four decimal places in the data.

The overall set is composed of 27 instances, and the results are reported in Table 2.

For each instance, Table 2 contains the same information of Table 1, but, obviously, the comparison is given with the CP algorithm by Pesant et al. (1998). For this algorithm, Table 2 reports the best solution found (UB), a column (Time) indicating either the computing time on a Sun SS1000 8xSuprSP if optimality is reached or a "–" otherwise, and the number of fails (Fails). Since the algorithm in Pesant et al. (1998) starts by "knowing" the best upper bound in the literature, we indicate with "$^\circ$" those solutions for which this upper bound has not been improved by the algorithm within the time limit of 1 day (neither an initial bound nor a primal heuristic is used by our algorithm). Again, the values indicated in the columns "UB" are proved to be optimal only if the time limit (1,800 CPU seconds for both version of our algorithm) has not been reached.

The behavior of our algorithm is, for these instances, very satisfactory. The AP-bound version is able to solve to optimality eight instances more than the CP approach in Pesant et al. (1998), and the Lagrangean-bound one is even better solving to optimality two more instances, namely, `rc207.1` and `rc206.0`. A fair comparison of the computing times is again difficult, however, if an instance is optimally solved by all the algorithms, the number of fails

Table 2: TSPTW **Symmetric** Instances (Computing Times in Seconds).

| | | Pesant et al. 1998 | | | | AP-bound | | | | | Lagrangean-bound | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| name | $n$ | UB | | Time | Fails | rLB | UB | | Time | Fails | rLB | UB | | Time | Fails |
| rc201.0 | 25 | 378.62 | * | 51.1 | 607 | 367.67 | 378.62 | * | 0.2 | 60 | 378.62 | 378.62 | * | 0.2 | 188 |
| rc201.1 | 28 | 374.70 | * | 311.0 | 4k | 256.14 | 374.70 | * | 1.8 | 942 | 346.72 | 374.70 | * | 4.7 | 2k |
| rc201.2 | 28 | 427.65 | * | 261.8 | 4k | 369.78 | 427.65 | * | 0.8 | 255 | 423.34 | 427.65 | * | 0.4 | 49 |
| rc201.3 | 19 | 232.54 | * | 6.5 | 77 | 216.81 | 232.54 | * | 0.1 | 24 | 217.84 | 232.54 | * | 0.1 | 24 |
| rc202.0 | 25 | 246.22 | * | 4136.0 | 32k | 180.36 | 246.22 | * | 8.5 | 3k | 241.91 | 246.22 | * | 1.0 | 221 |
| rc202.1 | 22 | 206.53 | * | 1803.9 | 19k | 146.79 | 206.53 | * | 3.8 | 1k | 179.17 | 206.53 | * | 10.4 | 4k |
| rc202.2 | 27 | 341.77 | * | 696.2 | 5k | 282.06 | 341.77 | * | 3.1 | 1k | 334.06 | 341.77 | * | 5.0 | 1k |
| rc202.3 | 26 | 367.85 | * | 11107.7 | 65k | 282.41 | 367.85 | * | 33.1 | 10k | 330.05 | 367.85 | * | 105.5 | 36k |
| rc203.0 | 35 | 384.8 | ∘ | – | – | 227.29 | 396.20 | | – | – | 298.26 | 388.71 | | – | – |
| rc203.1 | 37 | 357.3 | ∘ | – | – | 217.08 | 399.90 | | – | – | 301.41 | 377.31 | | – | – |
| rc203.2 | 28 | 337.6 | ∘ | – | – | 230.26 | 337.46 | * | 166.4 | 54k | 295.32 | 337.46 | * | 94.3 | 24k |
| rc204.0 | 32 | 221.5 | ∘ | – | – | 166.96 | 221.45 | * | 1295.1 | 312k | 214.36 | 221.45 | * | 352.8 | 79k |
| rc204.1 | 28 | 206.4 | ∘ | – | – | 169.36 | 205.37 | * | 993.5 | 284k | 203.08 | 205.37 | * | 3.4 | 705 |
| rc204.2 | 40 | 379.0 | ∘ | – | – | 245.17 | 430.32 | | – | – | 330.50 | 420.15 | | – | – |
| rc205.0 | 26 | 251.65 | * | 652.4 | 4k | 197.00 | 251.65 | * | 9.2 | 3k | 230.67 | 251.65 | * | 7.9 | 2k |
| rc205.1 | 22 | 271.22 | * | 128.1 | 1k | 240.54 | 271.22 | * | 0.2 | 59 | 270.44 | 271.22 | * | 0.3 | 74 |
| rc205.2 | 28 | 436.64 | | – | – | 294.67 | 434.69 | * | 1289.1 | 454k | 320.50 | 434.69 | * | 1409.4 | 586k |
| rc205.3 | 24 | 361.24 | * | 7026.2 | 90k | 261.50 | 361.24 | * | 4.7 | 2k | 329.96 | 361.24 | * | 5.1 | 1k |
| rc206.0 | 35 | 495.8 | ∘ | – | – | 348.42 | 485.23 | | – | – | 432.58 | 485.23 | * | 338.1 | 78k |
| rc206.1 | 33 | 334.8 | ∘ | – | – | 276.02 | 334.73 | * | 48.9 | 8k | 321.50 | 334.73 | * | 22.9 | 3k |
| rc206.2 | 32 | 335.37 | * | 67543.1 | 301k | 260.39 | 335.37 | * | 170.4 | 34k | 322.07 | 335.37 | * | 24.1 | 4k |
| rc207.0 | 37 | 436.69 | | – | – | 331.65 | 436.69 | * | 572.0 | 83k | 397.59 | 436.69 | * | 991.0 | 134k |
| rc207.1 | 33 | 396.36 | | – | – | 288.21 | 396.36 | | – | – | 374.36 | 396.36 | * | 321.7 | 50k |
| rc207.2 | 30 | 246.41 | * | 6790.6 | 62k | 200.28 | 246.41 | * | 299.8 | 79k | 238.16 | 246.41 | * | 15.1 | 3k |
| rc208.0 | 44 | 381.1 | ∘ | – | – | 251.04 | – | | – | – | 323.78 | – | | – | – |
| rc208.1 | 27 | 239.1 | ∘ | – | – | 179.44 | 239.04 | * | 242.6 | 77k | 229.71 | 239.04 | * | 34.2 | 8k |
| rc208.2 | 29 | 214.0 | ∘ | – | – | 165.85 | 213.92 | * | 13.9 | 4k | 213.92 | 213.92 | * | 1.4 | 246 |

gives a very good indication of the performance of the algorithms. Specifically, the number of fails of our algorithms is almost always much smaller than the one of the CP approach in Pesant et al. (1998), and in general the Lagrangean-bound version performs better than the AP-bound one. The only instance for which our algorithms are not able to find a feasible solution within the time limit is instance `rc208.0`. The fact that the Lagrangean-bound version performs clearly better than the AP-bound one is possibly motivated by the big improvement of the bound at the root node, which is often close to the optimal solution value.

In Pesant et al. (1998) the results for seven instances over the 27 in the benchmark are improved with respect to the state-of-the-art result obtained by effective heuristic approaches (Rochat and Taillard 1995 and Taillard et al. 1995). Besides these seven instances, our algorithms improve the results for four more instances, namely `rc203.2`, `rc205.2`, `rc206.0` and `rc208.2`.

# 9. Conclusions

We have presented a new algorithm for solving the Traveling Salesman Problem with Time Windows. The algorithm is based on a CP framework, and extensively uses problem-oriented propagation algorithms, and, in particular, a filtering algorithm based on the calculation of bounds on the optimal solution value. This filtering algorithm, which has been presented as a general framework (Focacci et al. 1999a), turns out to be affective for the TSPTW, and the overall approach is effective also if compared with the state-of-the-art methods in the literature.

A final remark concerns the use of the same algorithm without the cost-based filtering approach, i.e., a pure CP implementation. We have already shown in Focacci et al. (1999b) that the cost-based filtering algorithm is crucial for TSPTW: the number of instances of the previous sets solved to optimality are less than one third of those solved with the cost-based algorithm, and the computing times are at least ten times larger.

# Acknowledgments

We are grateful to Matteo Fischetti for suggesting the Lagrangean Relaxation technique, and for many fruitful discussions. We also thank Gilles Pesant for kindly providing us the Solomon instances, and Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel for providing us their code for SOPs separation. Thanks are due to anonymous referees for helpful comments that improved the paper.

# References

Applegate, D., R.E. Bixby, V. Chvátal, W. Cook. 1998. On the solution of traveling salesman problems. *Documenta Mathematica* **Extra Volume ICM III** 645-656.

Ascheuer, N. 1995. *Hamiltonian Path Problems in the On-line Optimization of Flexible Manufacturing Systems*. PhD thesis, Technische Universität Berlin, Berlin, Germany.

Ascheuer, N., M. Fischetti, M. Grötschel. 2000. A polyhedral study of the asymmetric travelling salesman problem with time Windows. *Networks* **36** 69-79.

Ascheuer, N., M. Fischetti, M. Grötschel. 2001. Solving asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming* **90** 475-506.

Baker, E.K. 1983. An exact algorithm for the time-constrained travelling salesman problem. *Operations Research* **31** 938-945.

Balas, E., M. Fischetti, W. Pulleyblank. 1995. The precedence constrained asymmetric travelling salesman problem. *Mathematical Programming* **68** 241-265.

Balas, E., N. Simonetti. 2001. Linear time dynamic programming algorithms for new classes of restricted TSPs: a computational study. *INFORMS Journal on Computing* **13** 56-75.

Baptiste, P., C. Le Pape, W. Nuijten. 1995. Incorporating efficient operations research algorithms in constraint-based scheduling. *Proceedings of the 1st International Joint Workshop on Artificial Intelligence and Operations Research*, Timberline, OR, June 6-10.

Benoist, T., F. Laburthe, B. Rottembourg. 2001. Lagrange-relaxation and constraint programming collaborative schemes for the travelling tournament problems. *Proceedings of the CP-AI-OR'01*, IC-PARC, Ashford, UK, April 8-10. `http://www.icparc.ic.ac.uk/cpAIOR01/program.htm`.

Carlier, J., E. Pinson. 1995. An algorithm for solving job shop scheduling. *Management Science* **35** 164-176.

Carpaneto, G., S. Martello, P. Toth. 1988. Algorithms and codes for the assignment problem. *Annals of Operations Research* **13** 193-223.

Caseau, Y., P. Koppstein. 1992. A rule-based approach to a time-constrained traveling salesman problem. *Proceedings of the $2^{nd}$ International Symposium of Artificial Intelligence and Mathematics*, Fort Lauderdale, FL.

Caseau, Y., F. Laburthe. 1994. Improved CLP scheduling with task intervals. P. Van Hentenryck, ed. *Logic Programming - Proceedings of the 1994 International Conference on Logic Programming.* MIT Press, Cambridge, MA. 369-383.

Caseau, Y., F. Laburthe. 1996. Cumulative scheduling with task intervals. *Proceedings of the JICSLP'96.* Bonn, Germany, September 2-6. 363-377.

Caseau, Y., F. Laburthe. 1997. Solving small TSPs with constraints. L. Naish, ed. *Logic Programming - Proceedings of the 1994 International Conference on Logic Programming.* MIT Press, Cambridge, MA. 316-330.

Christofides, N., A. Mingozzi, P. Toth. 1981. State space relaxation procedures for the computation of bounds to routing problems. *Networks* **11** 145-164.

Dell'Amico, M., S. Martello. 1997. Linear assignment. M. Dell'Amico, F. Maffioli, S. Martello, eds. *Annotated Bibliographies in Combinatorial Optimization*, Wiley, Chichester, UK. 355-371.

Desrochers, M., J. Desrosiers, M.M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40** 342-354.

Desrosiers, J. Y. Dumas, M.M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser, eds. *Network Routing.* Elsevier, Amsterdam, The Netherlands. 35-139.

Dumas, Y., J. Desrosiers, E. Gelinas, M.M. Solomon. 1995. An optimal algorithm for the travelling salesman problem with time windows. *Operations Research* **43** 367-371.

Fischetti, M., P. Toth. 1997. A polyhedral approach to the Asymmetric Traveling Salesman Problem. *Management Science* **43**, 1520-1536.

Focacci, F., A. Lodi, M. Milano. 1999a. Cost-based domain filtering. J. Jaffar, ed. *Principle and Practice of Constraint Programming - CP'99.* LNCS 1713. Springer-Verlag, Berlin, Germany. 189-203.

Focacci, F., A. Lodi, M. Milano. 1999b. Solving TSP with time windows with constraints. D. De Schreye, ed. *Logic Programming - Proceedings of the 1999 International Conference on Logic Programming.* MIT Press, Cambridge, MA. 515-529.

Focacci, F., A. Lodi, M. Milano. 2000. Cutting planes in constraint programming: a hybrid approach. R. Dechter, ed. *Principle and Practice of Constraint Programming - CP 2000.* LNCS 1894. Springer-Verlag, Berlin, Germany. 187-201.

Focacci, F., A. Lodi, M. Milano. 2002. Embedding relaxations in global constraints for solving TSP and TSPTW. *Annals of Mathematics and Artificial Intelligence* **34** 291-311.

Garey, M.R., D.S. Johnson. 1979. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, San Francisco, CA.

Grötschel, M., L. Lovász, A.J. Schrijver. 1988. *Geometric Algorithms and Combinatorial Optimization.* Wiley, New York.

ILOG. 2000a. *ILOG Scheduler 4.4* Reference Manual. Paris, France.

ILOG. 2000b. *ILOG Solver 4.4* Reference Manual. Paris, France.

Johnson, D.S., L.A. McGeoch. 1997. The traveling salesman problem: a case study. E. Aarts, J.K. Lenstra, eds. *Local Search in Combinatorial Optimization.* Wiley, Chichester, UK. 215-310.

Jünger, M., G. Rinaldi, S. Thienel. 2000. Practical performance of efficient minimum cut algorithms. *Algorithmica* **26** 172-195.
http://www.informatik.uni-koeln.de/ls_juenger/projects/mincut.html

Langevin, A., M. Desrochers, J. Desrosiers, F. Soumis. 1993. A two-commodity flow formulation for the traveling salesman and makespan problem with time windows. *Networks* **23** 631-640.

Mingozzi, A., L. Bianco, S. Ricciardelli. 1997. Dynamic programming strategies for the travelling salesman problem with time windows and precedence constraints. *Operations Research* **45** 365-377.

Padberg, M., G. Rinaldi. 1990. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming* **47** 19-36.

Pesant, G., M. Gendreau, J.-Y. Potvin, J.M. Rousseau. 1998. An exact constraint logic programming algorithm for the travelling salesman problem with time windows. *Transportation Science* **32** 12-29.

Pesant, G., M. Gendreau, J.-Y. Potvin, J.M. Rousseau. 1999. On the flexibility of constraint programming models: from single to multiple Time Windows for the travelling salesman problem. *European Journal of Operational Research* **117** 253-263.

Régin, J.C. 1994. A filtering algorithm for constraints of difference in CSPs. *Proceedings of the 12$^{th}$ National Conference on Artificial Intelligence - AAAI'94*. MIT Press, Cambridge, MA. 362-367.

Rochat Y., E.D. Taillard. 1995. Probabilistic diversification and intensification in local search for Vehicle Routing. *Journal of Heuristics* **1** 147-167.

Savelsberg, M.W.P. 1985. Local search in routing problems with time windows. *Annals of Operations Research* **4** 285-305.

Sellmann, M., T. Fahle. 2001. CP-based lagrangean relaxation for a multimedia application. *Proceedings of CP-AI-OR'01*, IC-PARC, Ashford, UK, April 8-10. `http://www.icparc.ic.ac.uk/cpAIOR01/program.htm`.

Solomon, M.M. 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research* **35** 254-265.

Taillard, E.D., P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin. 1995. A new neighborhood structure for the vehicle routing problems with time windows. Publication CRT-95-66, Centre de Recherche sur les Transports, Université de Montréal, Montréal, Quebec, Canada.