New State-Space Relaxations for Solving the Traveling Salesman Problem with Time Windows

Baldacci, Roberto; Mingozzi, Aristide; Roberti, Roberto

Published in:
I N F O R M S Journal on Computing

Link to article, DOI:
10.1287/ijoc.1110.0456

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

INFORMS Journal on Computing
Vol. 24, No. 3, Summer 2012, pp. 356–371
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1110.0456

New State-Space Relaxations for Solving the Traveling Salesman Problem with Time Windows

Roberto Baldacci
Department of Electronics, Computer Sciences and Systems (DEIS), University of Bologna, 40136 Bologna, Italy, r.baldacci@unibo.it

Aristide Mingozzi
Department of Mathematics, University of Bologna, 40126 Bologna, Italy, mingozzi@csr.unibo.it

Roberto Roberti
Department of Electronics, Computer Sciences and Systems (DEIS), University of Bologna, 40136 Bologna, Italy, roberto.roberti6@unibo.it

The traveling salesman problem with time windows (TSPTW) is the problem of finding in a weighted digraph a least-cost tour starting from a selected vertex, visiting each vertex of the graph exactly once according to a given time window, and returning to the starting vertex. This $\mathcal{NP}$-hard problem arises in routing and scheduling applications. This paper introduces a new tour relaxation, called *ngL*-tour, to compute a valid lower bound on the TSPTW obtained as the cost of a near-optimal dual solution of a problem that seeks a minimum-weight convex combination of nonnecessarily elementary tours. This problem is solved by column generation. The optimal integer TSPTW solution is computed with a dynamic programming algorithm that uses bounding functions based on different tour relaxations and the dual solution obtained. An extensive computational analysis on basically all TSPTW instances (involving up to 233 vertices) from the literature is reported. The results show that the proposed algorithm solves all but one instance and outperforms all exact methods published in the literature so far.

Key words: traveling salesman problem; time windows; state-space relaxations
History: Accepted by Michela Milano, Area Editor for Constraint Programming and Hybrid Optimization; received May 2010; revised November 2010, February 2011; accepted February 2011. Published online in *Articles in Advance* May 17, 2011.

1. Introduction

The *traveling salesman problem with time windows* (TSPTW) considered in this paper can be described as follows. A complete digraph $G = (V, A)$ is given, where $V = \{1, \ldots, n+2\}$ is a set of $n+2$ vertices and A is an arc set. Let $p, q \in V$ be two special vertices of G, and let $V' = V \setminus \{p, q\}$. Hereafter, we assume that $p = n+1$ and $q = n+2$. We indicate with $\Gamma_i \subseteq V$ the set of *successors* (i.e., $\Gamma_i = \{j \in V : (i, j) \in A\}$) and with $\Gamma_i^{-1} \subseteq V$ the set of *predecessors* (i.e., $\Gamma_i^{-1} = \{j \in V : (j, i) \in A\}$) of vertex $i \in V$ in G. We assume that $\Gamma_p^{-1} = \Gamma_q = \varnothing$ and that A does not contain arc (p, q).

A time window $[e_i, l_i]$ is associated with each vertex $i \in V$, where e_i and l_i represent the earliest and latest times to visit vertex i, respectively. A travel cost d_{ij} and a travel time t_{ij} are associated with each arc $(i, j) \in A$; the latter includes the service time at vertex i.

A salesman *tour* is defined as a path in G starting from vertex p at time e_p that visits each vertex $i \in V'$ within its time window and ends at vertex q before l_q. The salesman is allowed to arrive at a vertex $i \in V$ before time e_i, but, in this case, the service of vertex i is postponed until time e_i. Hereafter, we assume that graph G contains at least one salesman tour. The cost of a salesman tour is equal to the sum of the travel costs of the arcs traversed. The TSPTW consists of designing a tour of minimum total cost.

The problem is $\mathcal{NP}$-hard because it generalizes the classical traveling salesman problem (TSP). The TSPTW has practical applications in many routing and scheduling problems.

1.1. Literature Review

The first exact algorithms are owed to Christofides et al. (1981a) and Baker (1983). They proposed branch-and-bound methods to solve a variant of the problem where the total schedule time has to be minimized. The algorithm of Christofides et al. (1981a) was based on a technique called *state-space relaxation* (SSR), introduced by Christofides et al. (1981b), whereby the state space associated with a given dynamic programming (DP) recursion is relaxed into a space of smaller cardinality in such a way that the solution of the relaxed recursion provides a lower bound to the original problem. Christofides et al. (1981a) and Baker

(1983) reported solutions of TSPTW instances of up to 50 vertices.

Langevin et al. (1993) proposed a branch and bound using a two-commodity flow formulation and reported solutions of instances of up to 60 vertices.

DP-based algorithms have been proposed by Dumas et al. (1995), Mingozzi et al. (1997), Balas and Simonetti (2001), and Li (2009). Dumas et al. (1995) described a DP algorithm that makes use of sophisticated elimination tests aimed at reducing the state space and the number of state transitions. They reported the solutions of instances involving up to 200 vertices. Mingozzi et al. (1997) presented a DP algorithm for the TSP with time windows and precedence constraints based on the SSR technique, and the authors presented computational results for instances with up to 120 nodes. Balas and Simonetti (2001) considered the special case of the TSPTW where, for some initial ordering of the vertices, vertex i precedes vertex j if $j \geq i + k$ ($k > 0$), and the authors described a DP algorithm that is linear in n and exponential in k.

Recently, Li (2009) presented a DP algorithm based on a bi-directional resource-bounded label-correcting algorithm (see Righini and Salani 2006). Li (2009) reported the solutions of instances involving up to 233 vertices and solved a number of open instances to optimality.

Branch-and-cut methods for the TSPTW have been proposed by Ascheuer et al. (2001), and more recently, by Dash et al. (2012). Ascheuer et al. (2001) described heuristics and branch-and-cut methods to solve a real-world application (stacker crane optimization) that can be modeled as a TSPTW. The cutting plane algorithms described by Ascheuer et al. (2001) use three alternative integer programming formulations of the TSPTW and are based on the formulations and polyhedral analysis presented by the same authors (Ascheuer et al. 2000). They reported computational results for real-world instances with up to 233 vertices, showing that most TSPTW instances with up to 70 vertices can be solved to optimality by any of the three models. Dash et al. (2012) presented a new extended formulation for the TSPTW based on partitioning the time windows into subwindows called *buckets*. They described a branch-and-cut algorithm that makes use of the valid inequalities owed to Ascheuer et al. (2000, 2001) and new valid inequalities derived from the bucket formulation. Dash et al. (2012) reported the solution of instances with up to 233 vertices and solved a number of instances previously unsolved by Ascheuer et al. (2001).

Focacci et al. (2002) proposed a hybrid approach for solving the TSPTW, merging constraint programming algorithms and optimization techniques to compute bounds on the optimal solution value. Their algorithm solved instances involving up to 69 vertices.

Heuristic algorithms have been proposed, among others, by Gendreau et al. (1998), Wolfler Calvo (2000), Ohlmann and Thomas (2007), and López-Ibáñez and Blum (2010).

The TSPTW is a special case of the *vehicle routing problem with time windows* (VRPTW), where a fleet of identical vehicles located at a central depot have to be optimally routed to supply a set of customers with known demands so that each customer is visited exactly once within its time window. Because the TSPTW is a single-vehicle VRPTW, in general, any exact algorithm for the VRPTW can solve the TSPTW. Nonetheless, none of the state-of-the-art exact methods published in the literature for the VRPTW (see Jepsen et al. 2008, Desaulniers et al. 2008, Baldacci et al. 2011) has been proven to effectively solve the TSPTW.

These exact methods are based on the set partitioning (SP) formulation with additional cuts, where each column of the SP corresponds to a feasible route. Jepsen et al. (2008) and Desaulniers et al. (2008) derived branch-and-cut-and-price (BCP) algorithms, where the lower bound is computed by column-and-cut generation (CCG) methods using the simplex to solve the master problem and different procedures to solve the pricing problem. Such CCG methods based on the simplex are time consuming because the linear programming (LP) relaxation of the master problem is usually highly degenerate, and degeneracy implies alternative optimal dual solutions. Consequently, the generation of new columns and their associated variables may not change the value of the objective function of the master problem. Hence, the master problem may become large, and the overall method may become computationally slow.

Baldacci et al. (2011) proposed an exact method based on the SP model to solve the VRPTW as well as the *capacitated vehicle routing problem* (CVRP) that attempts to avoid the drawbacks of the CCG methods. This method consists of the following three main steps.

Step 1. Use dual-ascent procedures based on different nonelementary route relaxations, including a new route relaxation called *ng*-tour, to find a near-optimal dual solution of the LP relaxation of the SP model strengthened by valid inequalities. This step is fast and produces a lower bound for both the CVRP and the VRPTW that is, on average, above 99.0% of the optimal solution cost.

Step 2. Use a new CCG method based on the simplex to solve the LP relaxation of the SP model with additional cuts where the columns correspond to elementary routes. This method improves classical CCG by using the observation that any route of any optimal solution must have a reduced cost, with respect to the dual solution of Step 1, less than the gap between

a known upper bound and the lower bound achieved in Step 1. This observation is used in two ways: in generating the initial master problem and in solving the pricing problem to eliminate routes of negative reduced costs. Such an observation is sufficient in several cases to close the integrality gap. In practice, the resulting CCG method is faster than classical CCG methods because it requires fewer iterations to converge to an optimal dual solution.

Step 3. The optimal integer solution is obtained by solving, with an integer programming solver, a reduced SP containing every route whose reduced cost, with respect to the two dual solutions achieved by both Steps 1 and 2, is less than the gaps between a known upper bound and the lower bounds achieved by each step, respectively.

The computational results on both the CVRP and the VRPTW indicate that this exact method is significantly faster than the state-of-the-art BCP algorithms and can solve four out of the five open VRPTW instances.

Baldacci et al. (2010) provided deeper insights on the effectiveness of such an approach and showed that it can be adapted to solve several types of vehicle routing problems.

The method of Baldacci et al. (2011) cannot be used directly to solve the TSPTW because Step 2 of the bounding method would require the generation of feasible TSPTW solutions of negative reduced cost and Step 3 would require the generation of a subset of TSPTW solutions, including all optimal ones. However, Step 1 of the bounding method and the *ng*-tour relaxation can be extended to the TSPTW, as described below.

1.2. Contribution of This Paper

In this paper, we present a new exact method for the TSPTW that borrows Step 1 of the bounding method and the *ng*-tour relaxation from Baldacci et al. (2011) but differs from it substantially because our method uses a new state-space relaxation, called *ngL*-tour, which extends the *ng*-tour relaxation using the structural properties of the TSPTW. This new relaxation dominates the *ng*-tour relaxation and provides tighter lower bounds for hard TSPTW instances.

The optimal TSPTW solution is obtained by a simple iterative procedure calling, at each iteration, a DP algorithm that uses bounding functions and effective dominance rules to reduce the state-space graph. These bounding functions are based on a combination of the *ng*- and *ngL*-tour relaxations and the dual solution corresponding to the lower bound achieved.

We report extensive computational results on basically all TSPTW instances from the literature that are used by both exact and heuristic methods. The results show that the proposed algorithm solves all but one instance and outperforms all exact methods published in the literature so far.

The rest of this paper is organized as follows. In §2, we present the DP algorithm to solve the TSPTW. Section 3 describes three TSPTW relaxations, including the new *ngL*-tour relaxation. Section 4 presents the method to compute a valid lower bound, whereas in §5 we describe the bounding functions used to reduce the state-space graph. Section 6 reports the computational results. Finally, concluding remarks are given in §7.

2. An Exact Algorithm for the TSPTW

We assume that matrix t_{ij} satisfies the triangle inequality, so the time windows and the arc set A can be reduced using the reduction rules described by Dash et al. (2012).

We denote by $\hat{G} = (V, \hat{A})$ the precedence digraph where $(i, j) \in \hat{A}$ indicates that i has to precede j in any TSPTW solution. $\hat{A}$ is obtained as follows. First, we set $\hat{A} = \{(p, j) \in A\} \cup \{(j, q) \in A\}$, and we add arc (i, j) to $\hat{A}$ for any pair of vertices $i, j \in V$ such that $e_j + t_{ji} > l_i$. Furthermore, we add to $\hat{A}$ any arc $(i, k) \in A$ if there exists a vertex $j \in V \setminus \{p, i, k, q\}$ such that both arcs (i, j) and (j, k) belong to $\hat{A}$. This latter operation is repeated until no new arcs are added to $\hat{A}$.

We indicate with $\hat{\Gamma}_i \subseteq V$ the set of successors and with $\hat{\Gamma}_i^{-1} \subseteq V$ the set of predecessors of vertex $i \in V$ in graph $\hat{G}$. Arc $(i, j) \in \hat{A}$ is associated with the time $\hat{t}_{ij}$ computed as the cost of the shortest path in G from i to j using the travel time t_{ij} as the cost of arc $(i, j) \in A$. We assume $\hat{t}_{ii} = 0 \ \forall i \in V$.

Let us define a *forward path* $P = (p, i_1, \ldots, i_k = \sigma(P))$ as an elementary path starting from vertex p at time e_p, visiting vertices $V(P) = \{p, i_1, \ldots, i_k\}$ within their time windows, and ending at vertex $\sigma(P)$ at time $t(P)$ with $e_{\sigma(P)} \le t(P) \le l_{\sigma(P)}$. Let $\mathcal{P}(S, t, i)$ be the set of all forward paths visiting the subset of vertices $S \subseteq V$ and ending at vertex i at time t. We denote by $f(S, t, i)$ the cost of a least-cost path in the set $\mathcal{P}(S, t, i)$.

The cost z(TSPTW) of an optimal solution of the TSPTW is given by

$$z(\text{TSPTW}) = \min_{e_q \le t \le l_q} \{f(V, t, q)\}. \tag{1}$$

Functions $f(S, t, i)$ can be computed using DP as follows. Define the state set $\mathcal{S} = \{(S, t, i)\colon \forall S \subseteq V, \forall i \in S, e_i \le t \le l_i\}$. Let $\Omega(t, j, i)$ be the subset of departure times from vertex j to arrive at vertex i at time t, with $e_i \le t \le l_i$, when j is visited immediately before i. The set $\Omega(t, j, i)$ is defined as follows: (i) $\Omega(t, j, i) = \{t'\colon e_j \le t' \le \min\{l_j, t - t_{ji}\}\}$ if $t = e_i$ and (ii) $\Omega(t, j, i) = \{t - t_{ji}\colon e_j \le t - t_{ji} \le l_j\}$ if $e_i < t \le l_i$.

The DP recursion to compute functions $f(S, t, i)$ is as follows:

$$f(S, t, i) = \min_{(S', t', j) \in \Psi^{-1}(S,t,i)} \{f(S', t', j) + d_{ji}\} \quad \forall (S, t, i) \in \mathcal{S}, \quad (2)$$

where $\Psi^{-1}(S, t, i) = \{(S \backslash \{i\}, t', j)$: $\forall t' \in \Omega(t, j, i)$, $\forall j \in \Gamma_i^{-1} \cap S\}$.

The following initialization is required: $f(\{p\}, e_p, p) = 0$ and $f(\{p\}, t, p) = \infty$, $\forall t$, $e_p < t \leq l_p$. The size of the set $\mathcal{S}$ can be reduced by removing any state (S, t, i) that cannot lead to any feasible or optimal solution according to the following rules.

Dominance Rule 1. *Let (S, t, i) and (S, t', i) be two states such that $f(S, t, i) \geq f(S, t', i)$ and $t > t'$; then (S, t, i) is dominated by (S, t', i).*

Fathoming Rule 1. *Any state (S, t, i) cannot lead to any feasible solution if* (i) *there exists a vertex $j \in V \backslash S$ such that $t + \hat{t}_{ij} > l_j$ or* (ii) $\widehat{\Gamma}_i^{-1} \not\subset S$.

Fathoming Rule 2. *Let z_{ub} be an upper bound on the* TSPTW, *and let $b(S, t, i)$ be a lower bound on the cost of a least-cost path starting from $i \in S$ at time t, visiting each vertex in $V \backslash S$ within its time window, and finishing at vertex q before time l_q. Any state (S, t, i) such that*

$$f(S, t, i) + b(S, t, i) \geq z_{ub} \quad (3)$$

cannot lead to any optimal solution of cost smaller than z_{ub}.

In §5, we describe three methods to compute bounding functions $b(S, t, i)$ based on three SSRs of recursion (2) and on a valid lower bound LB described in §4.

To avoid the a priori computation of an upper bound z_{ub}, we use an iterative exact algorithm where, at each iteration h, recursion (2) is computed, and Fathoming Rule 2 is applied using an estimated upper bound z_{ub}^h. The exact algorithm is as follows.

Outline of the Exact Algorithm

Step 1. Compute the lower bound LB as described in §4, and set $z_{ub}^0 = LB$. Initialize $h = 1$, $\theta = \max\{\lceil 10^{-3} LB \rceil, 1\}$, and $z_{ub}^1 = \lceil LB \rceil + \theta$.

Step 2. Compute DP recursion (2) using the bounding functions $b(S, t, i)$ described in §5 and z_{ub}^h instead of z_{ub} in Fathoming Rule 2.

Step 3. We have two cases.

(a) No state (V, t, q), $e_q \leq t \leq l_q$, was generated. This happens if $z(\text{TSPTW}) > z_{ub}^h$. Set $h = h + 1$, $z_{ub}^h = z_{ub}^{h-1} + \theta$, go to Step 2.

(b) At least one of the states (V, t, q), $e_q \leq t \leq l_q$, was generated. The optimal TSPTW solution cost is given by $z(\text{TSPTW}) = \min_{e_q \leq t \leq l_q} \{f(V, t, q)\}$, stop.

In practice, the algorithm could terminate prematurely at iteration h while computing recursion (2) at Step 2, when the number of states generated exceeds the computer memory available. In this case, z_{ub}^{h-1} represents a valid lower bound on the TSPTW.

3. Relaxations of the TSPTW

In this section, we describe three relaxations of a TSPTW solution, called t-tour, ng-tour, and ngL-tour relaxations, that are used to compute valid lower bounds on the TSPTW and bounding functions $b(S, t, i)$ used by Fathoming Rule 2.

3.1. t-Tour Relaxation

The t-tour relaxation was introduced by Christofides et al. (1981b). This relaxation provides function $f(t, i)$, which corresponds to the cost of a least-cost non-necessarily elementary path, called (t, i)-path, starting from vertex p at time e_p, visiting a set of vertices (without two-vertex loops) within their time windows, and ending at vertex i at time $e_i \leq t \leq l_i$. The DP recursion to compute $f(t, i)$ is described in Christofides et al. (1981b). A t-tour is defined as the (t^*, q)-path such that $f(t^*, q) = \min_{e_q \leq t \leq l_q} \{f(t, q)\}$.

3.2. ng-Tour Relaxation

The t-path relaxation does not, in general, allow any detailed knowledge of the path of cost $f(t, i)$; hence one cannot impose additional conditions to ensure that such path provides a feasible solution to the original problem. To alleviate this drawback, Baldacci et al. (2011) introduced the ng-path relaxation. This consists of partitioning the set of all possible t-paths ending at vertex i at time t according to a mapping function that associates with each t-path a subset of the vertices visited, which depends on the order in which they are visited. The subset of vertices associated with each ng-path is used in the DP recursion to impose partial elementarity. The ng-path relaxation can be described in brief as follows.

Let $N_i \subseteq V$ be a set of selected vertices for vertex i such that $N_i \ni i$. The sets N_i allow us to associate with each forward path $P = (p, i_1, \ldots, i_k = \sigma(P))$ the subset $\Pi(P) \subseteq V(P)$ containing vertex i_k and every vertex i_r, $r = 1, \ldots, k-1$, of P that belongs to all sets $N_{i_{r+1}}, \ldots, N_{i_k}$ associated with the vertices $i_{r+1}, \ldots, i_k$ following i_r in P. Stated formally, the set $\Pi(P)$ is defined as

$$\Pi(P) = \left\{ i_r \colon i_r \in \bigcap_{s=r+1}^{k} N_{i_s},\ r = 1, \ldots, k-1 \right\} \cup \{i_k\}. \quad (4)$$

A *forward ng-path* (NG, t, i) is defined as a non-necessarily elementary path $P = (p, i_1, \ldots, i_{k-1}, i_k = i)$ starting from vertex p at time e_p, visiting a subset of vertices (even more than once) within their time windows such that $NG = \Pi(P)$, and ending at vertex i at time $e_i \leq t \leq l_i$ such that $i \notin \Pi(P')$, where $P' = (p, i_1, \ldots, i_{k-1})$ is an ng-path. It is worth mentioning that an ng-path can contain a loop $(i_\ell = j, i_{\ell+1}, \ldots, i_{\ell+s} = j)$ for $s \geq 2$ if and only if there exists at least one index k such that $1 \leq k \leq s-1$ and $j \notin N_{i_{\ell+k}}$.

We denote by $f(NG, t, i)$ the cost of a least-cost forward ng-path (NG, t, i). Any (NG, t, q)-path ending at vertex q at time $e_q \le t \le l_q$ is called an ng-tour.

A valid lower bound LB on the TSPTW is given by

$$LB = \min_{\substack{e_q \le t \le l_q,\\ NG \subseteq N_q \text{ with } NG \ni q}} \{f(NG, t, q)\} \le z(\text{TSPTW}). \quad (5)$$

Functions $f(NG, t, i)$ can be computed using DP as follows. Define the state set $\widetilde{\mathcal{S}} = \{(NG, t, i)\colon \forall i \in V, \forall t, e_i \le t \le l_i, \forall NG \subseteq N_i \text{ s.t. } NG \ni i\}$.

The DP recursion to compute $f(NG, t, i)$ is

$$f(NG, t, i) = \min_{(NG', t', j) \in \widetilde{\Psi}^{-1}(NG, t, i)} \{f(NG', t', j) + d_{ji}\} \quad \forall (NG, t, i) \in \widetilde{\mathcal{S}}, \quad (6)$$

where $\widetilde{\Psi}^{-1}(NG, t, i) = \{(NG', t', j)\colon \forall NG' \subseteq N_j \text{ s.t. } NG' \ni j \text{ and } NG' \cap N_i = NG \backslash \{i\}, \forall t' \in \Omega(t, j, i), \forall j \in \Gamma_i^{-1}\}$.

The following initialization is required: $f(\{p\}, e_p, p) = 0$ and $f(\{p\}, t, p) = \infty \ \forall t$ such that $e_p < t \le l_p$.

An optimal ng-tour is defined as the ng-path (NG^*, t^*, q) such that

$$f(NG^*, t^*, q) = \min_{(NG, t, q) \in \widetilde{\mathcal{S}}} \{f(NG, t, q)\}. \quad (7)$$

The size of the state set $\widetilde{\mathcal{S}}$ can be reduced by removing via the following dominance rule any state that cannot lead to an optimal ng-tour.

Dominance Rule 2. *Let (NG, t, i) and (NG', t', i) be two states of $\widetilde{\mathcal{S}}$ such that $f(NG, t, i) \le f(NG', t', i)$, where $NG \subseteq NG'$ and $t < t'$. State (NG', t', i) is dominated by (NG, t, i).*

Notice that a stronger version of Dominance Rule 2 can be obtained by replacing the condition $t < t'$ with $t \le t'$. However, we found it to be computationally convenient to apply Dominance Rule 2 instead of this stronger version.

3.2.1. Choosing the Sets N_i, $i \in V$. Lower bound LB, computed by expression (5), strongly depends on the sets N_i, $i \in V$. When $N_i = V$, $\forall i \in V$, then $f(NG, t, i)$ provides elementary least-cost paths, and the optimal ng-tour provided by expression (7) corresponds to an optimal TSPTW solution.

The computational results of §6 were obtained by defining the sets N_i, $i \in V$, as follows. We set $N_p = \{p\}$ and $N_q = \Gamma_q^{-1}$, and we define $T_i = \{j \in \Gamma_i \cap \Gamma_i^{-1}\colon t_{ij} = 0 \text{ or } t_{ji} = 0\} \cup \{i\}$, $\forall i \in V'$. Let Δ be a user-defined parameter. First, we set $N_i = T_i$, $\forall i \in V'$. Then, for each $i \in V'$ such that $|N_i| < \Delta$, we add to N_i the $(\Delta - |T_i|)$-nearest vertices to i of the set $V' \backslash T_i$.

Notice that recursion (6) allows the ng-path to contain two-vertex loops, which can be eliminated using the method described by Christofides et al. (1981b). Nonetheless, we do not implement this method because of the additional memory required and because, in practice, whenever sets N_i, $i \in V$, are defined as described above using $\Delta = 11$ or $\Delta = 13$, the resulting (NG, t, i)-paths rarely contain two-vertex loops.

3.2.2. Implementation Issues. To speed up the whole solution process, it is important to note how the DP recursion (6) is computed and, in particular, how Dominance Rule 2 is implemented. We decided to compute (6) using a forward DP recursion with the variable state t as the stage. At stage t, the order in which states are expanded is defined by a queue where states having time t are maintained.

To apply Dominance Rule 2, we use an additional function $F_t(NG, i)$ representing the cost of a least-cost state $(NG, t', i) \in \widehat{\mathcal{S}}$ with $t' < t$ (i.e., $F_t(NG, i) = \min_{e_i \le t' < t}\{f(NG, t', i)\colon (NG, t', i) \in \widehat{\mathcal{S}}\}$). Let $\mathcal{B}(NG, i) = \{NG' \subseteq NG\colon NG' \ni i\}$, $\forall i \in V$, $\forall NG \subseteq N_i$ with $NG \ni i$. In practice, because we use values of Δ less than or equal to 13, these sets can be computed by complete enumeration before starting the DP recursion. From the definitions of $F_t(NG, i)$ and $\mathcal{B}(NG, i)$, it follows that a state (NG, t, i) is dominated according to Dominance Rule 2 if

$$f(NG, t, i) \ge \min_{NG' \in \mathcal{B}(NG, i)} \{F_t(NG', i)\}. \quad (8)$$

Notice that function $F_t(NG, i)$ can be recursively computed as

$$F_t(NG, i) = \min\{F_{t-1}(NG, i), f(NG, t-1, i)\} \quad \forall i \in V, \ \forall NG \subseteq N_i, \ NG \ni i. \quad (9)$$

3.3. ngL-Tour Relaxation

In the specific case of the TSPTW, the ng-tour relaxation can be enhanced by forcing any ng-tour to visit each vertex of a selected subset of vertices exactly once while satisfying the precedence constraints imposed for such vertices by the precedence digraph $\hat{G}$.

Consider a path $L = (i_0 = p, i_1, \ldots, i_h = q)$ in $\hat{G}$ from vertex p to vertex q. Because any arc (i, j) of $\hat{G}$ implies that vertex i must be visited before vertex j, then the h arcs of path L decompose any TSPTW solution into h subpaths $P_{i_{k-1}i_k}$, $k = 1, \ldots, h$, where $P_{i_{k-1}i_k}$ corresponds to the subpath of the TSPTW solution starting from vertex i_{k-1}, visiting each vertex of a subset $S_k \subset V \backslash \{i_0, \ldots, i_h\}$ exactly once, and ending at vertex i_k. Thus, any TSPTW solution is made up of any collection of h subpaths $P_{i_{k-1}i_k}$, $k = 1, \ldots, h$, such that $\bigcup_{k=1}^{h} S_k = V \backslash V(L)$.

For a given path L of $\hat{G}$ as defined above, it is possible to force the ng-tour relaxation so that each vertex $i \in V(L)$ is visited exactly once as described in the following.

Let V_k be the subset of vertices that have to and/or can be visited in between i_{k-1} and i_k. For each $k = 1, \ldots, h$, we have

$$V_k = \{j \in V \backslash \{i_{k-1}\}: e_{i_{k-1}} + \hat{t}_{i_{k-1}j} \le l_j \text{ and } \max\{e_{i_{k-1}} + \hat{t}_{i_{k-1}j}, e_j\} + \hat{t}_{ji_k} \le l_{i_k}\}. \quad (10)$$

Notice that $V_k \cap V(L) = i_k$.

We define a *forward ngL(k)-path* as an *ng*-path (NG, t, i) ending at vertex $i \in V_k$ at time $e_i \le t \le l_i$, and with the additional restriction that each vertex $j \in \{i_0, i_1, \ldots, i_{k-1}\}$ is visited exactly once. An *ngL*-tour is defined as an *ngL*(h)-path (NG, t, q).

Let $\widetilde{\mathcal{S}}_k = \{(NG, t, i) \in \widetilde{\mathcal{S}}: i \in V_k\}$ be the subset of states corresponding to the forward *ngL*(k)-paths, and let $f_k(NG, t, i)$ be the cost of a least-cost *ngL*(k)-path $(NG, t, i) \in \widetilde{\mathcal{S}}_k$.

Functions $f_k(NG, t, i), k = 1, \ldots, h$ can be computed using the following iterative procedure. For each $k = 1, \ldots, h$, perform the following operations:

• Initialize $f_k(NG, t, i_{k-1}) = f_{k-1}(NG, t, i_{k-1})$, $\forall (NG, t, i_{k-1}) \in \widetilde{\mathcal{S}}_{k-1}$, and $f_k(NG, t, i) = \infty$, $\forall (NG, t, i) \in \widetilde{\mathcal{S}}_k$. We assume $f_0(\{p\}, e_p, p) = 0$ and $f_0(\{p\}, t, p) = \infty \ \forall t$ such that $e_p < t \le l_p$.

• Compute $f_k(NG, t, i)$ using the following DP recursion:

$$f_k(NG, t, i) = \min_{(NG', t', j) \in \widetilde{\Psi}^{-1}(NG, t, i)} \{f_k(NG', t', j) + d_{ji}\} \quad \forall (NG, t, i) \in \widetilde{\mathcal{S}}_k. \quad (11)$$

Because of the initialization and the definition of the sets V_k, $k = 1, \ldots, h$, any *ngL*(h)-path corresponding to $f_k(NG, t, i_k)$ visits every vertex of $V(L)$ exactly once.

An optimal *ngL*-tour is defined as the *ngL*(h)-path $(NG^*, t^*, q) \in \widetilde{\mathcal{S}}_h$ such that

$$f_h(NG^*, t^*, q) = \min_{(NG, t, i) \in \widetilde{\mathcal{S}}_h} \{f_h(NG, t, i)\}. \quad (12)$$

The value $f_h(NG^*, t^*, q^*)$ depends on the path L of $\hat{G}$ used. In our computational experiments, the path L was chosen as the path in $\hat{G}$ of maximal cardinality from vertex p to vertex q in order to maximize the number of vertices visited exactly once by any *ngL*-tour.

The following dominance rule, similar to Dominance Rule 2 described in §3.2 for the *ng*-tour relaxation, can be used to reduce the size of each state set $\widetilde{\mathcal{S}}_k$, $k = 1, \ldots, h$.

Dominance Rule 3. *Let* (NG, t, i) *and* (NG', t', i) *be two states of* $\widetilde{\mathcal{S}}_k$ *such that* $f_k(NG, t, i) \le f_k(NG', t', i)$, *where* $NG \subseteq NG'$ *and* $t < t'$. *State* (NG', t', i) *is dominated by* (NG, t, i) *and can be removed from* $\widetilde{\mathcal{S}}_k$.

4. A Valid Lower Bound LB on the TSPTW

In this section, we describe lower bounds on the TSPTW based on the three tour relaxations described in §3.

Let $H = \{1, 2, \ldots, h\}$ be the index set of the tours of a given tour relaxation of G (i.e., *t*-tour, *ng*-tour, or *ngL*-tour). Let R_k be the sequence of vertices visited by tour $k \in H$, and let δ_{ik} be the number of times vertex i is visited by tour $k \in H$. Let c_k be the cost of tour $k \in H$. Associate with each vertex $i \in V'$ a penalty $u_i \in \mathbb{R}$, and let $\mathbf{u} = (u_1, \ldots, u_n)$. For a given penalty vector $\mathbf{u}$, a valid lower bound $LR(\mathbf{u})$ on the TSPTW is given by

$$LR(\mathbf{u}) = \min_{k \in H} \left\{ c_k - \sum_{i \in V'} \delta_{ik} u_i \right\} + \sum_{i \in V'} u_i. \quad (13)$$

A better lower bound LB on the TSPTW can be computed using subgradient optimization to solve the following Lagrangean dual problem:

$$\text{(LD)} \quad z(LD) = \max_{u \in \mathbb{R}^n} \{LR(\mathbf{u})\}. \quad (14)$$

Solving problem LD by subgradient optimization can be time consuming because it might involve a large number of iterations where, at each iteration, an optimal value of problem $LR(\mathbf{u})$ for a given penalty vector $\mathbf{u}$ must be found. The optimal value of problem $LR(\mathbf{u})$ is computed with the DP recursion associated with the tour relaxation chosen, replacing arc costs d_{ij} with $d'_{ij} = d_{ij} - u_j$, $\forall (i, j) \in A \backslash A_q$, and $d'_{iq} = d_{iq}$, $\forall (i, q) \in A_q$, where $A_q = \{(i, q): i \in \Gamma_q^{-1}\}$.

Problem LD is equivalent to the following linear program D:

$$\text{(D)} \quad z(D) = \max \ w + \sum_{i \in V'} u_i \quad (15)$$

$$\text{s.t.} \ w \le c_k - \sum_{i \in V'} \delta_{ik} u_i \quad \forall k \in H, \quad (16)$$

$$w \in \mathbb{R} \quad \text{and} \quad u_i \in \mathbb{R}, \quad \forall i \in V'. \quad (17)$$

Dualizing D, we obtain the following problem F:

$$\text{(F)} \quad z(F) = \min \ \sum_{k \in H} c_k y_k \quad (18)$$

$$\text{s.t.} \ \sum_{k \in H} \delta_{ik} y_k = 1 \quad \forall i \in V', \quad (19)$$

$$\sum_{k \in H} y_k = 1, \quad (20)$$

$$y_k \ge 0 \quad \forall k \in H. \quad (21)$$

Problem F seeks a minimum-weight convex combination of tours such that each vertex in V' has, on average, a degree of 1. Notice that variables $u_i \in \mathbb{R}$, $\forall i \in V'$, and $w \in \mathbb{R}$ of problem D represent the duals of constraints (19) and (20), respectively. The optimal

solution cost of problem F is equal to the optimal solution cost of problem LD.

Problem F can be solved by column generation where, at each iteration, the tour of minimum negative reduced cost with respect to the dual solution of the current master problem must be computed. In our computational experiments, we found that this method of solving F requires fewer iterations than solving LD using subgradient optimization.

To solve F, we use a column generation procedure that differs from standard column generation methods because the master problem is not solved with the simplex but instead by using the dual-ascent heuristic proposed by Baldacci et al. (2011). Below, we describe how to initialize the master problem, the method used to solve the master problem, and the procedure used to solve the pricing problem.

4.1. Initializing the Master Problem

The master problem, called problem $\bar{\text{F}}$, corresponds to problem F where the tour set H is replaced by a small subset $\bar{H} \subseteq H$ of elementary but nonnecessarily Hamiltonian tours that are generated as follows. The procedure consists of executing a limited number of subgradient iterations to solve problem LD. The optimal tour found at each iteration is made elementary and is then added to $\bar{H}$. At the end, the procedure provides a lower bound LB_0 and a corresponding solution of problem D. The procedure can be described as follows.

Set $\bar{H} = \varnothing$ and $LB_0 = 0$. Initialize the Lagrangean penalties $\mu_i \in \mathbb{R}$, $i \in V'$, as $\mu_i = 0$. Perform *Maxit*0 iterations of the following subgradient method:

(a) Define $d'_{ij} = d_{ij} - \mu_j$, $\forall (i, j) \in A \backslash A_q$, and $d'_{iq} = d_{iq}$, $\forall (i, q) \in A_q$.

(b) Find the tour $R^* = (p, i_1, \ldots, i_h, q)$ (i.e., t-tour, ng-tour, or ngL-tour) of minimum cost $c'(R^*)$ with respect to the modified arc costs d'_{ij}. Let δ^*_j be the number of times that vertex $j \in V'$ is visited by the tour R^*.

(c) If $c'(R^*) + \sum_{i \in V'} \mu_i > LB_0$, update $LB_0 = c'(R^*) + \sum_{i \in V'} \mu_i$ and set $\mu^0_i = \mu_i$, $\forall i \in V'$.

(d) Add to $\bar{H}$ the elementary, but nonnecessarily Hamiltonian, tour R derived by removing from R^* any vertex i_r visited at position r, which has been previously visited at some position $k < r$.

(e) Because any feasible TSPTW solution must satisfy $\delta^*_j = 1$, $\forall j \in V'$, update penalties μ_i, $\forall i \in V'$ using the usual subgradient expressions, setting $z_{ub} = 1.2(c'(R^*) + \sum_{i \in V'} \mu_i)$ in computing the step size.

Notice that $(\mathbf{u}^0, w^0)$, where $\mathbf{u}^0 = (\mu^0_1, \ldots, \mu^0_n)$ and $w^0 = LB_0 - \sum_{i \in V'} \mu_i$, represents a feasible solution of problem D of cost LB_0.

4.2. Solving the Master Problem

Instead of solving problem $\bar{\text{F}}$ using the simplex, we use a dual-ascent heuristic based on the following theorem.

THEOREM 1. *Let $\boldsymbol{\lambda} = (\lambda_0, \lambda_1, \ldots, \lambda_n)$ be a vector of $n+1$ penalties, where penalties $\lambda_i \in \mathbb{R}$, $\forall i \in V'$, are associated with constraints* (19) *and where penalty λ_0 is associated with constraint* (20). *For each tour $k \in \bar{H}$, define $\lambda(R_k) = \sum_{i \in V'} \delta_{ik} \lambda_i + \lambda_0$ and $\delta(R_k) = \sum_{i \in V'} \delta_{ik}$. A feasible dual solution $(\mathbf{u}, w)$ of $\bar{F}$ of cost $z(\bar{F}(\boldsymbol{\lambda})) = w + \sum_{i \in V'} u_i$ is obtained as follows*:

$$\begin{aligned} u_i &= \min_{k \in \bar{H}_i} \left\{ \frac{\delta_{ik}(c_k - \lambda(R_k))}{\delta(R_k)} \right\} + \lambda_i, \quad i \in V', \\ w &= \lambda_0, \end{aligned} \tag{22}$$

where $\bar{H}_i \subseteq \bar{H}$ is the subset of tours covering vertex $i \in V'$.

PROOF . See Baldacci et al. (2011). □

To find a near-optimal dual solution $(\mathbf{u}, w)$ of problem $\bar{\text{F}}$, we perform an a priori defined number, *Maxit*1, of subgradient iterations to modify the penalty vector $\boldsymbol{\lambda}$. Baldacci et al. (2011) have shown that a valid subgradient of function $z(\bar{F}(\boldsymbol{\lambda}))$ at point $\boldsymbol{\lambda}$ can be computed by associating with the current dual solution $(\mathbf{u}, w)$ of $\bar{\text{F}}$ a nonnecessarily feasible solution $\mathbf{y}$ of $\bar{F}$, such that $z(\bar{F}(\boldsymbol{\lambda})) = \sum_{k \in \bar{H}} c_k y_k$, which is defined as follows.

Let $\widetilde{H}$ be the index set of the distinct tours producing u_i in expressions (22), and let $k(i)$ be the index of the tour in $\widetilde{H}$ associated with u_i, $i \in V'$. Define variables $\xi^i_k \in \{0, 1\}$, $k \in \widetilde{H}$, $i \in V'$, as $\xi^i_{k(i)} = 1$ and $\xi^i_k = 0$, $\forall k \in \widetilde{H} \backslash \{k(i)\}$, $\forall i \in V'$. The solution $\mathbf{y}$ is computed as $y_k = 0$, $\forall k \in \bar{H} \backslash \widetilde{H}$, and $y_k = \sum_{i \in V'} \delta_{ik} / \delta(R_k) \xi^i_k$, $\forall k \in \widetilde{H}$.

Let α_i, $\forall i \in V'$, and α_0 denote the values of the left-hand side of constraints (19) and (20) with respect to solution $\mathbf{y}$, respectively. The values of α_i, $\forall i \in V'$, and α_0 are used to update the penalty vector $\boldsymbol{\lambda}$ by means of the usual subgradient expressions, where we use $z_{ub} = 1.2 z(\bar{F}(\boldsymbol{\lambda}))$. We denote by $(\bar{\mathbf{u}}, \bar{w})$ the final dual solution of $\bar{\text{F}}$ of cost $\bar{z}$ achieved by the procedure previously described.

4.3. Solving the Pricing Problem

The pricing problem consists of finding the tour of minimum reduced cost with respect to the current dual solution $(\bar{\mathbf{u}}, \bar{w})$ of the master problem $\bar{\text{F}}$. Define the modified arc cost $\bar{d}_{ij}$ as follows:

$$\bar{d}_{ij} = \begin{cases} d_{ij} - \bar{u}_j & \forall (i, j) \in A \backslash A_q, \\ d_{iq} - \bar{w} & \forall (i, q) \in A_q. \end{cases} \tag{23}$$

It is quite easy to see that the reduced cost $\bar{c}_k = c_k - \sum_{i \in V'} \delta_{ik} \bar{u}_i - \bar{w}$ of tour $k \in H$ with respect to $(\bar{\mathbf{u}}, \bar{w})$ is equal to the cost of the tour (i.e., t-tour, ng-tour, or ngL-tour) using the modified arc costs $\bar{d}_{ij}$ instead of d_{ij}. Therefore, to compute the tour of minimum reduced cost with respect to $(\bar{\mathbf{u}}, \bar{w})$, it is sufficient to compute the associated DP recursion using $\bar{d}_{ij}$ instead of d_{ij}.

5. Computing Bounding Functions $b(S, t, i)$

In this section, we describe how to compute bounding functions $b(S, t, i)$, introduced in §2, to reduce the state-space graph associated with the DP recursion (2).

We define a *backward path* $\bar{P} = (\sigma(\bar{P}) = i_k, i_{k+1}, \ldots, i_h, q)$ as a path starting from vertex $\sigma(\bar{P})$ at time $t(\bar{P})$, visiting each vertex in $V(\bar{P}) = \{i_k, i_{k+1}, \ldots, i_h, q\}$ within its time window, and ending at vertex q before time l_q.

Consider the forward path P associated with $f(S, t, i)$. The least-cost TSPTW solution containing P is obtained by adding to P a least-cost backward path $\bar{P}$ of cost $c(\bar{P})$ starting from i at time $t' \geq t$ and visiting all vertices $V \backslash S$.

The method to compute functions $b(S, t, i)$ is based on the following proposition. For a given dual solution $(\mathbf{u}^*, w^*)$ of problem F providing lower bound LB, define the modified arc costs d^*_{ij} as

$$d^*_{ij} = \begin{cases} d_{ij} - u^*_j & \forall (i, j) \in A \backslash A_q, \\ d_{iq} & \forall (i, q) \in A_q. \end{cases} \tag{24}$$

Proposition 1. *Let $lb(S, t, i)$ be a lower bound on the cost, using arc costs d^*_{ij} instead of d_{ij}, of any backward path starting from vertex i at time $t' \geq t$, with $e_i \leq t' \leq l_i$. A valid lower bound on the cost $c(\bar{P})$ of a least-cost backward path $\bar{P}$ starting from vertex i at time $t' \geq t$ and visiting vertices $V \backslash S$ is given by*

$$b(S, t, i) = lb(S, t, i) + \sum_{j \in V' \backslash S} u^*_i. \tag{25}$$

Proof. Let $c^*(\bar{P})$ be the cost of $\bar{P}$ using arc costs d^*_{ij}. From the definition of $lb(S, t, i)$, we have

$$lb(S, t, i) \leq c^*(\bar{P}). \tag{26}$$

Because $\bar{P}$ visits exactly once each vertex $j \in V' \backslash S$, then

$$c^*(\bar{P}) = c(\bar{P}) - \sum_{j \in V' \backslash S} u^*_i. \tag{27}$$

From expressions (25)–(27) we obtain

$$b(S, t, i) \leq c(\bar{P}). \quad \square \tag{28}$$

In the following, we describe three methods to compute the lower bound $lb(S, t, i)$ on $c(\bar{P})$ that are based on the three relaxations described in §3.

A *backward* (t, i)-path is a nonnecessarily elementary path $\bar{P}$ starting from i at time t, visiting a subset of vertices (once or more) within their time windows, and ending at vertex q before l_q. We denote by $f^{-1}(t, i)$ the cost of a least-cost backward (t, i)-path.

A *backward ng*-path (NG, t, i) is a nonnecessarily elementary path $\bar{P}$ starting from i at time t, visiting a subset of vertices (once or more) within their time windows such that $NG = \Pi^{-1}(\bar{P})$, and ending at vertex q before l_q such that $i \notin \Pi^{-1}(P')$, where $P' = (i_{k+1}, \ldots, i_h, i_q)$ is a backward *ng*-path, and the set $\Pi^{-1}(\bar{P})$ is defined as

$$\Pi^{-1}(\bar{P}) = \{i_k\} \cup \left\{ i_r \colon i_r \in \bigcap_{s=k}^{r-1} N_{i_s},\ r = k+1, \ldots, h \right\}. \tag{29}$$

We denote by $f^{-1}(NG, t, i)$ the cost of a least-cost backward *ng*-path (NG, t, i).

For a given longest path $L = (i_0 = p, i_1, \ldots, i_k, i_{k+1}, \ldots, i_h, i_{h+1} = q)$, let $\bar{V}_k$ be the subset of vertices that have to and/or can be visited in between i_k and i_{k+1}, $k = 0, \ldots, h$; that is,

$$\begin{aligned} \bar{V}_k = \{ & j \in V \backslash \{i_{k+1}\} \colon e_{i_k} + \hat{t}_{i_k j} \leq l_j \text{ and} \\ & \max\{e_{i_k} + \hat{t}_{i_k j}, e_j\} + \hat{t}_{j i_{k+1}} \leq l_{i_{k+1}} \}. \end{aligned} \tag{30}$$

A *backward ngL(k)*-path (NG, t, i) is a backward *ng*-path starting from vertex $i \in \bar{V}_k$ that visits the vertices $\{i_{k+1}, \ldots, i_h, i_q\}$ exactly once. We denote by $f^{-1}_k(NG, t, i)$ the cost of a least-cost backward *ngL(k)*-path.

Functions $f^{-1}(t, i)$, $f^{-1}(NG, t, i)$, and $f^{-1}_k(NG, t, i)$ can be computed with the same DP recursions used to compute $f(t, i)$, $f(NG, t, i)$, and $f(NG, k, t, i)$, respectively, on the TSPTW instances that result from the following operations: (i) for each vertex $i \in V$, replace time window $[e_i, l_i]$ with $[e'_i, l'_i]$, where $e'_i = l_q - l_i$ and $l'_i = l_q - e_i$; and (ii) replace the cost and time matrices $[d_{ij}]$ and $[t_{ij}]$ with their transposed matrices, $[d_{ij}]^T$ and $[t_{ij}]^T$.

Let functions $f^{-1}(t, i)$, $f^{-1}(NG, t, i)$, and $f^{-1}_k(NG, t, i)$ be computed by replacing the arc costs d_{ij} with d^*_{ij}. These functions allow for the computing of the lower bound $lb(S, t, i)$ as follows:

(a) From functions $f^{-1}(t, i)$, we derive $lb(S, t, i) = \min_{t' \geq t}\{f^{-1}(t', i)\}$.

(b) From functions $f^{-1}(NG, t, i)$, we derive $lb(S, t, i) = \min_{\substack{NG \subseteq N_i \text{ s.t.} \\ NG \cap S = \{i\},\ t' \geq t}} \{f^{-1}(NG, t', i)\}$.

(c) From functions $f^{-1}_k(NG, t, i)$, we derive $lb(S, t, i) = \min_{\substack{NG \subseteq N_i \text{ s.t.} \\ NG \cap S = \{i\},\ t' \geq t}} \{f^{-1}_{k'}(NG, t', i)\}$, where k' is defined as follows:

(i) $i \notin V(L)$: k' is the index of the first vertex $i_{k'}$ of the longest path L such that $i_{k'} \notin S$, and

(ii) $i \in V(L)$: k' is the index of the vertex of L such that $i_{k'} = i$.

6. Computational Results

This section reports on the computational experiments of our algorithm, which was implemented in C and compiled with Visual Studio 2008, 32-bit professional edition. The runs were performed on a Sony Vaio P8400 laptop (Intel Core 2 Duo, 2.26 GHz) equipped with 4 GB of RAM, running under Windows Vista, 32-bit home edition.

We denote by BMR.*ng* the exact algorithm of §2 using either the *ng*-tour or the *ngL*-tour relaxations, according to the criterion described in §6.1, to compute the lower bound and the bounding functions $b(S, t, i)$. We denote by BMR.*t* the exact method when the *t*-tour relaxation is used to compute the lower bound and the bounding functions.

We tested both BMR.*ng* and BMR.*t* on the seven classes of instances available at http://iridia.ulb.ac.be/∼manuel/tsptw-instances and the class proposed by Mingozzi et al. (1997). The classes proposed by Langevin et al. (1993), Dumas et al. (1995), and Mingozzi et al. (1997) were easily solved (the most difficult instance was solved in 12 seconds by BMR.*ng*), so relative results are omitted. The results for the other five classes are reported next and are compared with the following state-of-the-art exact methods:

- The branch-and-cut method of Ascheuer et al. (2001) (hereafter, AFG): runs were performed on a Sun SPARC Station 10 with a time limit (*tl*) of five hours (i.e., $tl = 18{,}000$ seconds).
- The constraint programming-based method of Focacci et al. (2002) (hereafter, FLM): runs were performed on a PC Pentium III at 700 MHz with 128 MB of RAM ($tl = 1{,}800$ seconds). The results reported in the tables are, on each instance, the best achieved by the two versions of their method (i.e., *AP-bound* and *Lagrangean-bound*).
- The branch-and-cut method of Dash et al. (2012) (hereafter, DGLT): runs were performed on a workstation Intel at 2.40 GHz running SUSE Linux 10.1 ($tl = 18{,}000$ seconds);
- The DP algorithm of Li (2009) (hereafter, LI): runs were performed on a Lenovo Thinkstation with four Intel processors at 2 GHz, with 4 GB of RAM and a Linux operating system (no time limit imposed).

These exact methods were not tested on all five instance classes. In the tables we present here, "*tl*" indicates that the time limit was reached. According to SPEC (http://www.spec.org/benchmarks.html), our machine is 10% slower than the machine used by Dash et al. (2012) and 10% faster than the machine used by Li (2009). No benchmarks are available to compare our machine with the ones used by Ascheuer et al. (2001) and Focacci et al. (2002). Nevertheless, our machine is clearly faster (say, 6–10 times faster) than these two machines.

6.1. Parameter Setting

To find a parameter setting for BMR.*ng* to use on all instances, we performed preliminary tests by applying both *ng*-tour and *ngL*-tour relaxations and varying parameter Δ.

In Table 1, we present such results on a selected set of representative instances. The columns report the instance name; the number of vertices of graph G ($|V|$); the length of the longest path L of the precedence graph $\hat{G}$ ($|L|$); the percentage of precedences (%*Prec*), computed as $\%Prec = 100|\hat{A}|/((n+2)(n+1)/2)$; and the optimal solution cost (z^*). Then, we indicate the tour relaxation used (*Rel*), which can be either *ng*-tour or *ngL*-tour, and the setting of parameter Δ. Finally, the lower bound achieved (*LB*) is reported with the percentage %*LB* ($= 100LB/z^*$), the time (in seconds) to compute the lower bound (T_{LB}), the cardinality (in thousands) of the set $\mathcal{S}$ generated by the exact algorithm of §2 ($|\mathcal{S}|$), and the total CPU time in seconds (T_{TOT}).

Table 1 shows that increasing Δ lets us compute better lower bounds and generate smaller state-space graphs when solving the problem to optimality. Nonetheless, as the complexity of computing recursions (6) and (11) and testing Fathoming Rule 2

Table 1 Results of BMR.*ng* on a Representative Set of Instances

Instance	$\lvert V\rvert$	$\lvert L\rvert$	%*Prec*	z^*	*Rel*	Δ	*LB*	%*LB*	T_{LB}	$\lvert\mathcal{S}\rvert$	T_{TOT}
n150w120.002	152	13	77.8	677	*ng*-tour	11	663.3	97.98	69.7	—	—
					ng-tour	13	663.8	98.05	124.8	—	—
					ngL-tour	11	668.0	98.66	77.2	14,473	211.3
					ngL-tour	13	668.4	98.73	130.5	12,149	260.6
n200w120.005	202	14	79.6	840	*ng*-tour	11	827.5	98.51	155.0	—	—
					ng-tour	13	828.9	98.68	195.1	—	—
					ngL-tour	11	831.5	98.99	204.9	—	—
					ngL-tour	13	833.2	99.19	237.4	29,072	574.3
rbg152.3	152	12	81.3	1,539	*ng*-tour	11	1,536.2	99.82	47.3	22,015	192.5
					ng-tour	13	1,536.7	99.85	64.7	17,387	206.7
					ngL-tour	11	1,537.7	99.92	73.3	17,191	201.9
					ngL-tour	13	1,538.2	99.95	81.5	14,151	213.2
rbg233.2	233	22	90.9	2,188	*ng*-tour	11	2,187.0	99.95	72.1	9,200	124.8
					ng-tour	13	2,187.0	99.95	92.2	8,906	136.9
					ngL-tour	11	2,187.0	99.95	231.1	6,750	364.3
					ngL-tour	13	2,187.0	99.95	253.3	5,265	359.6

increases, the CPU time to compute the lower bound and the total CPU time may increase as well.

The *ngL*-tour relaxation dominates the *ng*-relaxation but is more time consuming. At the same time, the lower bound that can be achieved with the *ngL*-tour relaxation is sometimes just slightly better than or the same as that which can be achieved with the *ng*-tour relaxation. In particular, we can see that the higher the %*Prec* is, the lower the increment of the lower bound is when applying *ngL*-tour relaxation instead of *ng*-tour relaxation.

Notice that *ngL*-tour relaxation allows us to solve a few instances that cannot be solved with *ng*-tour relaxation. Thus, BMR.*ng* uses *ng*-tour relaxation if $\%Prec \geq 80$ and *ngL*-tour relaxation otherwise.

In computing the lower bound *LB* (see §4.3), Δ was set equal to 11 when using *ng*-tour relaxation and equal to 13 when using *ngL*-tour relaxation. In computing the bounding functions $b(S, t, i)$ (see §5), Δ was set equal to 13 for both relaxations.

Finally, in the column generation method described in §4, for both BMR.*ng* and BMR.*t* we set $Maxit0 = 200$ and $Maxit1 = 150$. In the exact method, because of memory limits, $|\mathcal{S}|$ is limited to 10^8. Both BMR.*ng* and BMR.*t* terminate prematurely when this limit is reached. Notice that if this limit is reached at iteration h, z_{ub}^{h-1} represents a valid lower bound on the optimal solution cost (see §2).

6.2. Ascheuer Instances

The Ascheuer class consists of 50 asymmetric real-world instances with up to 233 vertices. Both travel costs and times are integer values. Travel times satisfy the triangle inequality.

Like Dash et al. (2012), we present the results on this class after dividing the 50 instances into 32 easy instances (those solved by Ascheuer et al. 2001 within the time limit of five hours) and the remaining 18 hard instances.

Table 2 presents the computational results on easy instances. The column labels are the same as those

Table 2 Results on Easy Ascheuer Instances

			BMR.*ng*						BMR.*t*		AFG		FLM		DGLT		LI
Instance	$\|V\|$	z^*	%*Prec*	*LB*	%*LB*	T_{LB}	$\|\mathcal{S}\|$	T_{TOT}	%*LB*	T_{TOT}	%*LB*	T_{TOT}	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}
rbg010a	12	149	72.7	149.0	100.0	0.1		0.1	99.5	0.8	99.3	0.1	99.3	0.0	100.0	0.0	0.1
rbg017	17	148	72.1	148.0	100.0	0.1		0.1	97.2	0.7	100.0	0.8	93.2	0.1	99.3	0.0	0.1
rbg017.2	17	107	41.2	107.0	100.0	0.1		0.1	100.0	0.6	100.0	0.0	93.5	0.0	100.0	0.0	19.9
rbg016a	18	179	83.0	179.0	100.0	0.1		0.1	100.0	2.2	98.9	0.2	93.3	0.1	100.0	0.0	0.1
rbg016b	18	142	56.9	142.0	100.0	0.2		0.2	95.9	4.8	93.7	8.8	88.7	0.1	97.2	0.2	0.1
rbg017a	19	146	49.1	146.0	100.0	0.3		0.3	100.0	0.4	100.0	0.1	100.0	0.1	100.0	0.0	10.3
rbg019a	21	217	91.9	217.0	100.0	0.2		0.2	100.0	0.4	100.0	0.0	100.0	0.0	100.0	0.0	0.1
rbg019b	21	182	61.4	182.0	100.0	0.3		0.3	99.4	2.8	98.9	54.5	96.2	0.2	99.5	0.3	0.2
rbg019c	21	190	45.7	190.0	100.0	0.5		0.5	98.5	1.4	95.8	8.7	94.2	0.3	96.8	0.9	35.3
rbg019d	21	344	78.1	344.0	100.0	0.3		0.3	98.8	2.6	99.7	0.7	98.3	0.0	100.0	0.2	0.1
rbg021	21	190	45.7	190.0	100.0	0.5		0.5	98.6	2.5	95.8	8.7	94.2	0.3	96.8	0.9	35.5
rbg021.2	21	182	43.3	182.0	100.0	0.4		0.4	100.0	0.8	100.0	0.2	98.4	0.2	100.0	0.1	48.0
rbg021.3	21	182	42.9	182.0	100.0	1.4		1.4	98.9	3.0	97.8	27.1	91.2	0.4	98.4	2.7	101.2
rbg021.4	21	179	40.5	179.0	100.0	1.2	1	1.4	99.7	2.2	98.9	5.8	92.7	0.3	100.0	0.2	692.8
rbg021.5	21	169	39.5	169.0	100.0	1.1		1.1	99.6	1.9	98.8	6.6	94.7	0.2	100.0	0.3	1,320.5
rbg021.6	21	134	24.3	134.0	100.0	0.6	1	0.7	100.0	1.4	99.3	1.3	95.5	0.7	100.0	0.3	7,876.4
rbg021.7	21	133	19.5	133.0	100.0	1.5	1	1.7	98.5	2.1	96.2	4.3	94.0	0.6	100.0	0.6	—
rbg021.8	21	132	18.6	132.0	100.0	1.6	15	1.8	98.4	2.3	97.7	17.4	97.0	0.6	98.5	2.8	—
rbg021.9	21	132	18.6	132.0	100.0	3.1	18	3.3	98.0	2.2	97.0	26.1	94.7	0.8	98.5	2.8	—
rbg020a	22	210	53.7	210.0	100.0	0.7		0.7	100.0	0.9	100.0	0.2	98.6	0.0	100.0	0.0	10.5
rbg027a	29	268	43.1	267.9	100.0	1.3	1	1.5	99.5	7.3	99.3	2.2	97.0	0.2	99.3	1.4	0.7
rbg031a	33	328	75.8	328.0	100.0	0.5		0.5	99.1	7.4	100.0	1.7	97.3	2.7	100.0	0.2	0.1
rbg033a	35	433	77.6	433.0	100.0	0.5		0.5	99.3	6.5	100.0	1.8	92.8	1.0	99.8	0.9	0.2
rbg034a	36	403	72.4	402.9	100.0	3.2	5	3.5	99.2	9.4	99.5	0.9	97.0	55.2	100.0	1.9	1.0
rbg035a	37	254	76.3	254.0	100.0	0.6		0.6	98.8	12.7	100.0	1.8	87.0	3.5	100.0	0.2	0.1
rbg035a.2	37	166	70.9	166.0	100.0	23.5	6	24.1	98.4	20.1	95.2	64.8	93.4	36.8	100.0	5.3	—
rbg038a	40	466	78.1	466.0	100.0	2.0		2.0	100.0	6.2	100.0	4,232.2	100.0	0.2	100.0	1.5	0.5
rbg040a	42	386	79.2	384.9	99.7	1.3	1	1.5	97.7	5.7	92.0	751.8	73.1	738.1	96.6	3.6	0.8
rbg050a	52	414	44.2	414.0	100.0	16.1	4	16.6	99.7	24.4	100.0	18.6	98.8	95.6	100.0	24.5	—
rbg055a	57	814	84.9	814.0	100.0	1.8	1	2.1	99.3	7.3	99.9	6.4	98.2	2.5	100.0	3.5	2.4
rbg067a	69	1,048	88.8	1,048.0	100.0	1.9	1	2.2	99.4	17.6	99.9	5.9	98.6	4.0	100.0	3.6	3.2
rbg125a	127	1,409	92.8	1,409.0	100.0	5.6	2	6.4	99.3	71.0	99.5	229.8	98.2	tl	100.0	9.6	3.5
Avg					99.99	2.3		2.4	99.08	7.2	98.53	171.5	94.96	30.5	99.40	2.1	376.4
Solved								32		32		32		31		32	27

Table 3 Results on Hard Ascheuer Instances

			BMR.*ng*						BMR.*t*		AFG	FLM		DGLT		LI
Instance	$\|V\|$	z^*	%*Prec*	*LB*	%*LB*	T_{LB}	$\|\mathscr{S}\|$	T_{TOT}	%*LB*	T_{TOT}	%*LB*	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}
rbg041a	43	402	77.3	402.0	100.0	2.0		2.0	95.4	4.2	89.8	87.6	*tl*	97.5	146.8	2.2
rbg042a	44	411	71.6	411.0	100.0	6.6		6.6	97.7	5.0	95.9	90.0	149.8	98.3	188.3	35.6
rbg048a	50	487	47.8	487.0	100.0	12.3		12.3	100.0	14.7	93.2	91.2	*tl*	100.0	129.2	—
rbg049a	51	**484**	63.4	472.1	97.5	19.5	6	20.4	95.7	13.7	84.3	83.3	*tl*	95.9	*tl*	—
rbg050b	52	**512**	61.1	504.8	98.6	19.1	2	19.9	96.5	8.1	87.3	87.1	*tl*	96.3	*tl*	—
rbg050c	52	**526**	49.8	522.4	99.3	20.3	93	21.7	98.5	23.1	96.4	95.8	*tl*	98.3	*tl*	—
rbg086a	88	1,051	91.7	1,050.0	99.9	2.1	1	2.5	99.1	14.5	99.1	97.3	*tl*	99.8	4.9	1.9
rbg092a	94	1,093	89.8	1,091.6	99.9	6.7	3	7.3	99.7	14.2	99.2	98.8	*tl*	99.8	90.4	9.0
rbg132.2	132	1,083	88.2	1,080.2	99.7	45.5	294	47.9	99.4	80.5	97.2	n.a.		99.8	2,761.1	—
rbg132	132	1,360	94.2	1,360.0	100.0	10.6	1	11.3	98.9	48.9	97.3	n.a.		99.5	37.6	6.1
rbg152.3	152	1,539	81.3	1,536.2	99.8	47.3	20,971	166.9	99.4	238.8	98.8	n.a.		99.9	10,353.3	—
rbg152	152	1,783	94.0	1,783.0	100.0	18.8	3	19.8	99.3	36.8	98.7	n.a.		99.8	43.7	11.5
rbg172a	174	1,799	93.9	1,799.0	100.0	37.1	4	38.4	99.2	455.7	98.8	n.a.		99.8	425.5	544.3
rbg193.2	193	**2,017**	89.1	2,015.3	99.9	50.8	414	54.4	99.6	76.3	97.6	n.a.		99.7	*tl*	—
rbg193	193	2,414	94.5	2,414.0	100.0	46.4	16	47.9	99.8	65.9	98.8	n.a.		100.0	159.6	3,165.7
rbg201a	203	2,189	94.8	2,189.0	100.0	54.3	4	56.0	99.3	716.2	98.6	n.a.		99.9	462.7	581.1
rbg233.2	233	**2,188**	90.9	2,187.0	100.0	72.1	8,969	120.6	99.8	124.4	98.1	n.a.		99.7	*tl*	—
rbg233	233	2,689	95.4	2,688.0	100.0	78.9	35	81.0	99.5	319.3	98.0	n.a.		100.0	749.4	1,433.3
Avg					99.70	30.6		40.9	98.72	125.6	95.95	91.39	149.8	99.11	1,196.3	579.0
Solved								18		18			1		13	10

Notes. Optimal values in bold represent instances solved for the first time by BMR.*ng*. n.a., data not available.

Table 4 Results on Pesant Instances

			BMR.*ng*						BMR.*t*		FLM		DGLT		LI
Instance	$\|V\|$	z^*	%*Prec*	*LB*	%*LB*	T_{LB}	$\|\mathscr{S}\|$	T_{TOT}	%*LB*	T_{TOT}	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}
rc201.0	27	378.62	84.3	378.62	100.0	0.1		0.1	100.0	0.1	100.0	0.2	100.0	0.0	0.1
rc201.1	30	374.70	85.3	374.70	100.0	0.2		0.2	99.0	0.3	68.4	1.8	100.0	0.0	0.1
rc201.2	30	427.65	83.7	427.65	100.0	0.2		0.2	100.0	0.1	99.0	0.4	100.0	0.0	0.1
rc201.3	21	232.54	82.9	232.54	100.0	0.1		0.1	99.9	0.1	93.7	0.1	98.7	0.1	0.1
rc202.0	27	246.22	45.3	246.22	100.0	0.4		0.4	100.0	0.2	98.2	1.0	100.0	0.2	63.0
rc202.1	24	206.53	55.1	206.53	100.0	0.4		0.4	94.8	0.4	71.1	3.8	100.0	0.1	1.1
rc202.2	29	341.77	61.3	341.77	100.0	0.2		0.2	94.6	0.8	82.5	3.1	100.0	0.1	0.4
rc202.3	28	367.85	50.5	367.85	100.0	0.3		0.3	97.9	0.6	76.8	33.1	100.0	0.2	5.1
rc203.0	37	377.45	24.0	376.22	99.7	4.2	2	4.6	83.4	31.0	79.0	*tl*	95.4	3,437.7	—
rc203.1	39	356.99	25.1	356.66	99.9	14.1	1	14.5	89.1	20.4	84.4	*tl*	97.6	2,722.7	—
rc203.2	30	337.47	42.3	337.47	100.0	0.5		0.5	94.1	2.3	87.5	94.3	100.0	0.9	392.7
rc204.0	34	221.45	16.8	221.45	100.0	1.7		1.7	93.1	2.0	96.8	352.8	100.0	2.8	—
rc204.1	30	205.37	14.7	205.37	100.0	4.3		4.3	92.3	9.0	98.9	3.4	99.0	14.9	—
rc204.2	42	**378.40**	14.6	374.77	99.0	33.5	7	36.4	92.3	29.4	87.3	*tl*	96.6	*tl*	—
rc205.0	28	251.65	60.1	251.65	100.0	0.3		0.3	100.0	1.4	91.7	7.9	100.0	0.1	1.9
rc205.1	24	271.22	67.8	271.22	100.0	0.2		0.2	98.9	0.2	88.7	0.2	100.0	0.0	0.4
rc205.2	30	434.70	68.5	434.70	100.0	0.4		0.4	92.7	1.2	67.8	1,289.1	100.0	0.3	0.3
rc205.3	26	361.24	64.6	361.24	100.0	0.2		0.2	100.0	0.2	72.4	4.7	100.0	0.0	0.1
rc206.0	37	485.23	62.6	485.23	100.0	0.8		0.8	90.3	1.7	89.1	338.1	97.5	73.3	1.6
rc206.1	35	334.73	57.8	334.73	100.0	1.0		1.0	96.3	0.8	96.0	22.9	98.2	85.0	42.5
rc206.2	34	335.37	55.8	335.37	100.0	0.9		0.9	98.7	0.9	96.0	24.1	98.8	84.8	189.5
rc207.0	38	436.69	38.7	436.69	100.0	0.6		0.6	95.1	2.0	75.9	572.0	99.5	19.0	18,691.8
rc207.1	35	396.36	37.8	396.36	100.0	1.4		1.4	94.4	1.6	94.4	321.7	98.4	34.9	10,864.5
rc207.2	32	246.41	33.3	246.41	100.0	0.6		0.6	98.4	0.7	96.7	15.1	97.2	116.4	—
rc208.0	46	**380.56**	8.6	380.56	100.0	38.1		38.1	92.8	63.4	85.1	*tl*	93.4	*tl*	—
rc208.1	29	239.04	13.5	239.04	100.0	6.9		6.9	96.5	0.9	96.1	34.2	96.1	990.6	—
rc208.2	31	213.92	12.7	213.92	100.0	1.3		1.3	94.4	1.2	100.0	1.4	100.0	7.9	—
Avg					99.95	4.2		4.3	95.51	6.4	87.91	135.9	98.75	303.7	1,680.8
Solved								27		27		23		25	18

Note. Optimal values in bold represent instances solved for the first time by BMR.*ng*.

used as in Table 1. For AFG, FLM, and DGLT, column "%*LB*" reports the percentage deviation of the lower bound at the root node of the decision tree. The last two lines of the table indicate averages on the percentage deviation of the lower bounds, the CPU times of the instances solved to optimality, and the number of instances solved by each method.

Table 2 shows that the lower bound computed by BMR.*ng* closes the gap on all but three instances. BMR.*ng* solves all 32 instances with the same performance of DGLT.

Table 3 reports the results on hard Ascheuer instances. In column "z^*" the optimal value is in bold when instances were solved for the first time by BMR.*ng*.

Table 3 shows that BMR.*ng* solves the five open instances and outperforms the other exact methods.

6.3. Pesant Instances

The Pesant class consists of 27 symmetric instances proposed by Pesant et al. (1998) with 21 to 46 vertices. Travel costs and times are the Euclidean distances truncated to four decimal places. Travel times include service times and satisfy the triangle inequality.

Table 4 reports the results on Pesant instances. The lower bounds computed by BMR.*ng* are of excellent quality. BMR.*ng* and BMR.*t* solve all instances of this class and clearly outperform FLM, DGLT, and LI.

6.4. Potvin Instances

The Potvin class consists of 28 symmetric instances introduced by Potvin and Bengio (1996) with 15 to 47 vertices (we neglect instances with fewer than 10 vertices). The instances feature $x - y$ coordinates. Travel times are Euclidean and include the service time at the starting vertex of each arc. Travel costs and travel times coincide.

In Table 5, we compare BMR.*ng* and BMR.*t* with LI. In column "z^*," the optimal solution cost is rounded to two decimal places. The last column indicates the CPU time of LI (a dash indicates when the instance was not solved).

From Table 5, both BMR.*ng* and BMR.*t* compare favorably with LI. BMR.*ng* is clearly superior to both BMR.*t* and LI because it solves all instances.

6.5. Gendreau Instances

The Gendreau class is made up of 28 groups of five instances proposed by Gendreau et al. (1998). Each

Table 5 Results on Potvin Instances

			BMR.*ng*						BMR.*t*		LI
Instance	$\|V\|$	z^*	%*Prec*	*LB*	%*LB*	T_{LB}	$\|\mathcal{S}\|$	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}
rc201.1	21	444.54	80.0	444.54	100.0	0.1		0.1	99.6	0.7	0.0
rc201.2	27	711.54	86.0	711.54	100.0	0.2		0.2	100.0	1.3	0.1
rc201.3	33	790.61	86.7	790.61	100.0	0.2		0.2	98.7	4.0	0.1
rc201.4	27	793.64	86.0	793.64	100.0	0.2		0.2	99.4	3.3	0.0
rc202.1	34	771.78	52.2	771.64	100.0	1.0	1	1.3	92.9	9.0	164.5
rc202.2	15	304.14	33.3	304.14	100.0	0.2		0.2	95.5	3.4	0.6
rc202.3	30	837.72	76.6	835.87	99.8	0.2	1	0.4	97.2	5.9	0.0
rc202.4	29	793.03	49.8	791.54	99.8	5.1	1	5.3	98.1	7.2	94.8
rc203.1	20	453.48	40.5	453.48	100.0	0.2		0.2	99.4	3.2	2.4
rc203.2	34	**784.16**	32.4	781.64	99.7	1.7	1	2.1	93.4	11.1	—
rc203.3	38	**817.53**	33.1	810.46	99.1	9.3	10	9.9	94.9	71.6	—
rc203.4	16	314.29	28.3	314.29	100.0	0.4		0.4	100.0	0.8	11.1
rc204.1	47	**878.64**	14.8	872.62	99.3	42.6	10	43.8	93.2	581.6[a]	—
rc204.2	34	**662.16**	16.2	650.94	98.3	21.3	151	23.1	90.5	473.4[a]	—
rc204.3	25	**455.03**	17.0	455.03	100.0	9.9		9.9	95.0	9.0	—
rc205.1	15	343.21	67.6	343.21	100.0	0.1		0.1	98.3	4.8	0.0
rc205.2	28	755.93	72.0	755.93	100.0	0.3		0.3	98.8	3.7	0.2
rc205.3	36	825.06	57.3	825.06	100.0	2.2		2.2	97.7	8.5	197.7
rc205.4	29	760.47	76.1	756.95	99.5	0.2	1	0.4	97.5	4.1	0.1
rc206.2	38	828.06	59.9	826.66	99.8	6.4	1	6.8	95.3	9.7	20.5
rc206.3	26	574.42	52.3	574.42	100.0	0.6		0.6	90.0	6.3	5.5
rc206.4	39	831.67	59.0	827.54	99.5	1.8	1	2.2	90.5	13.4	69.6
rc207.1	35	**732.68**	42.9	731.57	99.8	9.0	1	9.3	93.3	11.9	—
rc207.2	32	**701.25**	33.7	694.22	99.0	53.1	5	54.8	89.4	47.2	—
rc207.3	34	**682.40**	30.8	677.23	99.2	1.9	1	2.3	91.8	15.6	—
rc208.1	39	**789.25**	10.4	785.69	99.5	46.7	2	47.5	90.0	435.9[a]	—
rc208.2	30	**533.78**	13.1	533.78	100.0	30.1		30.1	93.8	25.2	—
rc208.3	37	**634.44**	10.7	622.48	98.1	82.0	24	86.8	94.1	532.6[a]	—
Avg					99.67	11.7		12.2	95.30	11.7	33.3
Solved								28		24	17

Note. Optimal values in bold represent instances solved for the first time by BMR.*ng*.
[a]BMR.*t* runs out of memory.

Table 6 Results on Gendreau Instances

		BMR.*ng*			BMR.*t*		LI			BMR.*ng*			BMR.*t*		LI
Instance	z^*	%*Prec*	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}	Instance	z^*	%Prec	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}
n20w120.001	267	62.8	100.0	0.4	100.0	0.2	0.2	n60w200.001	**410**	47.2	100.0	14.7	89.2	714.0[b]	—
n20w120.002	218	52.4	100.0	0.5	96.6	0.2	0.3	n60w200.002	**414**	52.1	100.0	7.4	[a]		—
n20w120.003	303	63.2	100.0	0.5	94.2	0.2	0.1	n60w200.003	**455**	46.1	97.6	192.1	[a]		—
n20w120.004	300	63.2	100.0	0.3	97.2	0.1	0.1	n60w200.004	**431**	52.2	99.7	18.4	96.6	14.7	—
n20w120.005	240	60.2	100.0	0.3	99.3	0.3	0.3	n60w200.005	**427**	47.5	99.6	20.0	[a]		—
n20w140.001	176	50.2	100.0	0.4	95.3	0.6	0.5	n80w100.001	**541**	74.2	100.0	3.8	[a]		—
n20w140.002	272	56.7	100.0	0.4	95.9	0.3	0.1	n80w100.002	**567**	80.4	100.0	1.1	[a]		—
n20w140.003	236	54.5	100.0	0.3	92.9	0.5	0.1	n80w100.003	**578**	75.7	100.0	3.4	[a]		—
n20w140.004	255	51.1	100.0	1.2	96.3	0.5	0.3	n80w100.004	**648**	82.0	99.6	20.4	98.3	1.2	—
n20w140.005	225	58.9	100.0	0.3	88.4	0.6	0.1	n80w100.005	**532**	76.4	100.0	6.1	[a]		—
n20w160.001	241	51.9	100.0	0.6	92.5	0.4	1.6	n80w120.001	**498**	70.8	99.9	27.0	[a]		—
n20w160.002	201	52.8	100.0	0.3	95.0	0.2	0.1	n80w120.002	**577**	73.6	99.9	16.0	[a]		—
n20w160.003	201	60.2	100.0	0.3	98.5	0.3	0.1	n80w120.003	**540**	73.1	100.0	6.4	[a]		—
n20w160.004	203	42.0	100.0	0.7	90.8	0.4	3.4	n80w120.004	**501**	68.7	98.1	61.4	[a]		—
n20w160.005	245	50.6	100.0	0.3	87.7	0.7	0.3	n80w120.005	**591**	72.2	99.3	39.6	[a]		—
n20w180.001	253	51.5	100.0	0.3	97.7	0.3	0.4	n80w140.001	**511**	68.0	100.0	7.5	[a]		—
n20w180.002	265	50.2	100.0	0.3	99.4	0.4	0.3	n80w140.002	**470**	67.8	99.8	77.8	[a]		—
n20w180.003	271	44.2	100.0	1.0	94.1	0.5	0.4	n80w140.003	**580**	68.7	99.1	46.5	[a]		—
n20w180.004	201	42.9	100.0	0.4	96.4	0.3	3.2	n80w140.004	**422**	67.8	99.6	38.8	[a]		—
n20w180.005	193	35.9	100.0	0.3	91.7	0.7	27.2	n80w140.005	**545**	68.4	100.0	8.1	94.5	282.7	—
n20w200.001	233	38.1	100.0	1.1	93.1	0.4	12.0	n80w160.001	**506**	62.6	100.0	93.7	[a]		—
n20w200.002	203	31.6	100.0	1.1	94.9	0.6	55.7	n80w160.002	**548**	60.6	99.9	194.8	[a]		—
n20w200.003	249	40.3	100.0	0.8	89.5	0.8	4.5	n80w160.003	**521**	60.4	100.0	58.2	[a]		—
n20w200.004	293	41.6	98.8	2.4	90.6	0.8	3.0	n80w160.004	**509**	63.5	100.0	50.8	[a]		—
n20w200.005	227	34.2	100.0	0.4	99.5	0.3	26.4	n80w160.005	**438**	63.3	100.0	58.1	92.8	576.8[b]	—
n40w120.001	434	70.4	100.0	2.1	[a]		15.7	n80w180.001	**551**	57.8	99.9	75.4	91.9	702.3[b]	—
n40w120.002	444	66.7	100.0	1.9	95.4	1.7	51.5	n80w180.002	**478**	58.5	100.0	61.5	[a]		—
n40w120.003	357	65.9	100.0	2.1	98.8	1.5	39.1	n80w180.003	**524**	55.3	100.0	84.3	89.7	967.6[b]	—
n40w120.004	303	63.8	100.0	1.9	88.7	2.4	28.6	n80w180.004	**479**	59.6	100.0	47.4	[a]		—
n40w120.005	350	68.3	100.0	1.3	98.9	0.9	2.4	n80w180.005	**470**	61.5	100.0	49.0	91.9	602.8[b]	—
n40w140.001	328	65.2	100.0	2.0	97.3	1.0	42.9	n80w200.001	**490**	47.5	99.3	177.4	[a]		—
n40w140.002	383	63.8	100.0	1.5	92.4	3.2	83.2	n80w200.002	**488**	55.1	99.9	32.1	[a]		—
n40w140.003	398	66.9	100.0	2.2	[a]		5.5	n80w200.003	**464**	50.0	99.6	39.5	[a]		—
n40w140.004	342	59.9	100.0	1.7	93.1	2.1	230.5	n80w200.004	**526**	54.1	99.1	63.3	[a]		—
n40w140.005	371	56.3	100.0	6.2	95.7	1.5	1,288.1	n80w200.005	**439**	49.0	99.8	46.9	[a]		—
n40w160.001	348	56.8	100.0	1.4	94.3	1.0	117.3	n100w80.001	**670**	84.3	100.0	4.5	[a]		—
n40w160.002	337	56.1	100.0	1.2	95.7	0.9	1,331.4	n100w80.002	**666**	86.4	99.9	2.5	98.3	2.9	—
n40w160.003	346	53.7	100.0	3.5	[a]		474.4	n100w80.003	**691**	85.4	100.0	1.4	97.9	7.0	—
n40w160.004	**288**	47.5	100.0	12.9	90.6	196.6	—	n100w80.004	**700**	84.1	100.0	2.8	[a]		—
n40w160.005	315	51.5	100.0	6.9	92.6	4.5	9,369.0	n100w80.005	**603**	82.3	100.0	2.6	[a]		—
n40w180.001	**337**	51.3	100.0	25.9	87.7	9.7	—	n100w100.001	**643**	79.7	100.0	18.4	[a]		—
n40w180.002	**347**	47.5	100.0	14.9	[a]		—	n100w100.002	**618**	81.0	100.0	2.5	96.3	37.3	—
n40w180.003	**279**	51.7	100.0	1.6	[a]		—	n100w100.003	**685**	80.5	100.0	4.5	96.3	15.8	—
n40w180.004	**354**	52.7	99.6	7.5	96.6	1.1	—	n100w100.004	**684**	79.9	99.5	83.6	[a]		—
n40w180.005	**335**	43.9	100.0	4.8	86.2	48.0	—	n100w100.005	**572**	78.2	100.0	18.6	[a]		—
n40w200.001	**330**	39.7	100.0	8.8	[a]		—	n100w120.001	**629**	78.2	100.0	34.5	[a]		—
n40w200.002	**303**	39.6	99.7	17.1	[a]		—	n100w120.002	**540**	77.4	99.2	24.1	[a]		—
n40w200.003	**339**	42.0	95.0	25.8	[a]		—	n100w120.003	**615**	74.2	100.0	61.6	97.1	33.2	—
n40w200.004	**301**	47.6	99.7	12.9	87.4	12.5	—	n100w120.004	**662**	79.7	99.7	91.4	[a]		—
n40w200.005	**296**	35.0	100.0	8.9	93.0	4.6	—	n100w120.005	**537**	74.9	100.0	42.1	[a]		—
n60w120.001	384	69.7	100.0	17.6	96.1	3.0	8,431.4	n100w140.001	**603**	71.2	100.0	38.1	[a]		—
n60w120.002	426	72.6	100.0	4.1	99.0	0.9	419.3	n100w140.002	**613**	74.3	100.0	44.4	[a]		—
n60w120.003	407	69.1	100.0	6.0	[a]		20,411.3	n100w140.003	**481**	73.9	100.0	51.5	[a]		—
n60w120.004	490	70.1	100.0	4.8	96.5	5.0	13,702.2	n100w140.004	**533**	74.3	100.0	24.7	[a]		—
n60w120.005	547	70.7	100.0	9.1	96.9	1.9	1,455.1	n100w140.005	**509**	73.1	99.1	66.9	90.5	725.4[b]	—

group consists of instances having the same number of vertices and the same time window width. The instances are obtained from the instances of Dumas et al. (1995), extending the time windows, which results in time windows ranging from 80 to 200 time units in increments of 20. The number of vertices ranges from 22 to 102.

The name of each instance indicates the number of vertices (excluding p and q), the width of the time windows, and the instance number in the

Table 6 (Continued)

		BMR.*ng*			BMR.*t*		LI			BMR.*ng*			BMR.*t*		LI
Instance	z^*	%*Prec*	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}	Instance	z^*	%*Prec*	%*LB*	T_{TOT}	%*LB*	T_{TOT}	T_{TOT}
n60w140.001	**423**	65.8	99.6	8.4	[a]		—	n100w160.001	**582**	69.4	100.0	117.4	92.0	310.0[b]	—
n60w140.002	462	67.3	99.8	11.5	96.0	2.8	4,497.5	n100w160.002	**530**	68.8	99.8	99.5	[a]		—
n60w140.003	**427**	66.0	100.0	11.8	94.6	10.1	—	n100w160.003	**495**	71.5	100.0	52.6	[a]		—
n60w140.004	**488**	67.6	100.0	14.2	93.7	2.9	—	n100w160.004	**580**	69.8	100.0	116.6	[a]		—
n60w140.005	460	66.1	99.0	33.4	90.6	8.5	5,148.5	n100w160.005	**586**	71.9	99.9	72.9	[a]		—
n60w160.001	**560**	63.5	99.3	42.6	90.9	458.2	—	n100w180.001	**568**	60.6	99.4	170.4	[a]		—
n60w160.002	**423**	62.3	100.0	7.7	98.7	2.2	—	n100w180.002	**503**	60.8	100.0	40.4	[a]		—
n60w160.003	**434**	54.5	99.7	34.8	92.8	184.0[b]	—	n100w180.003	**574**	61.9	99.0	135.2	[a]		—
n60w160.004	**401**	62.7	100.0	4.0	96.5	6.2	—	n100w180.004	**526**	61.7	100.0	128.4	[a]		—
n60w160.005	**501**	63.0	100.0	7.2	91.6	356.0	—	n100w180.005	**501**	60.0	98.6	117.9	[a]		—
n60w180.001	**411**	56.7	99.1	103.9	[a]		—	n100w200.001	**549**	55.7	100.0	424.3	[a]		—
n60w180.002	**399**	53.7	100.0	8.4	95.4	13.3	—	n100w200.002	**502**	55.9	100.0	86.9	[a]		—
n60w180.003	**444**	54.7	99.1	28.5	90.9	624.0[b]	—	n100w200.003	**557**	57.9	99.5	264.6	[a]		—
n60w180.004	**456**	54.9	99.9	51.6	93.5	20.8	—	n100w200.004	**521**	57.3	99.4	199.5	[a]		—
n60w180.005	**395**	50.3	98.4	39.3	[a]		—	n100w200.005	**486**	55.5	99.8	104.7	[a]		—
Avg											99.78	36.7	94.23	24.9	1,462.7
Solved												140		64	46

Note. Optimal values in bold represent instances solved for the first time by BMR.*ng*.
[a]Not attempted because $t_{ij} = 0$ for some $(i, j) \in A$.
[b]BMR.*t* runs out of memory.

group (e.g., n20w120.001 is the first instance of the group of five instances with 22 vertices and time windows of 120 units). In some papers, two groups of instances (n100w80.x and n100w100.x) are included in the Dumas class instead of the Gendreau class; nonetheless, such instances coincide. Two other groups (n100w180.x and n100w200.x) were not available on the Web at the time of this writing but were sent to us by Ohlmann and Thomas (2010).

Travel times and costs coincide, and they are first computed as truncated integer Euclidean distances and then modified to satisfy triangle inequality by iteratively setting $c_{ij} = t_{ij} = t_{ik} + t_{kj}$, if $t_{ik} + t_{kj} < t_{ij}$, until no violation is identified.

Because the *t*-tour relaxation requires strictly positive travel times, BMR.*t* was not tested on the Gendreau instances such that $t_{ij} = 0$ for some $(i, j) \in A$.

In Table 6, we report detailed computational results on Gendreau instances. BMR.*ng* and BMR.*t* are compared with LI. Table 6 shows that BMR.*ng* is much faster than LI and solves all 140 instances, whereas LI solves 46 of them. BMR.*t* solves 18 more instances than LI. The comparison of BMR.*t* and LI on the instances solved by both methods shows that BMR.*t* outperforms LI.

6.6. Ohlmann Instances

The Ohlmann class was proposed by Ohlmann and Thomas (2007) and consists of 25 instances divided into five groups. The instances have 152 or 202 vertices and are obtained from the Dumas instances by extending the time windows by 100 time units. Travel times and travel costs are computed as for the Gendreau class.

To our knowledge, no exact method has been tested on this class so far. In Table 7, we report the computational results of BMR.*ng*. BMR.*t* was not tested on these instances because the requirement of strictly positive travel times is not satisfied by any of the instances.

Table 7 Results on Ohlmann Instances

		BMR.*ng*					
Instance	z^*	%*Prec*	*LB*	%*LB*	T_{LB}	$\lvert\mathcal{S}\rvert$	T_{TOT}
n150w120.001	**732**	77.5	725.5	99.1	243.8	243	248.6
n150w120.002	**677**	77.8	668.4	98.7	130.5	12,149	260.6
n150w120.003	**747**	76.5	746.4	99.9	160.0	2	163.2
n150w120.004	**762**	78.0	761.6	99.9	172.8	1	176.0
n150w120.005	**689**	76.9	684.7	99.4	113.1	373	118.9
n150w140.001	**762**	72.5	754.0	98.9	198.2	14,993	350.2
n150w140.002	**753**	74.5	752.0	99.9	439.0	7	442.1
n150w140.003	**613**	73.3	608.5	99.3	420.6	1,671	440.9
n150w140.004	**676**	73.7	676.0	100.0	104.8		104.8
n150w140.005	**663**	73.1	662.0	99.8	365.9	3	369.4
n150w160.001	**704**	70.4	701.4	99.6	262.9	48	266.5
n150w160.002	**711**	69.3	709.7	99.8	213.1	5	216.4
n150w160.003	**608**	71.9	603.2	99.2	483.0	352	491.8
n150w160.004	**672**	69.9	672.0	100.0	570.6	12	577.1
n150w160.005	**658**	71.0	655.0	99.5	169.5	44	173.4
n200w120.001	**795**	78.3	793.3	99.8	303.8	57	308.8
n200w120.002	**721**	80.2	713.9	99.0	110.4	20,223	257.1
n200w120.003	**879**	80.4	868.6	98.8	151.1	57,625	958.6
n200w120.004	**777**	79.1	775.8	99.8	412.3	13	417.1
n200w120.005	**840**	79.6	833.2	99.2	237.4	29,072	574.3
n200w140.001	**830**	77.1	826.2	99.5	1,217.8	6,261	1,274.9
n200w140.002	**760**	76.3	756.2	99.5	614.5	407	624.5
n200w140.003	**758**	76.3	756.0	99.7	250.7	133	259.0
n200w140.004	816[a]	75.3	807.1	98.9	592.9	—	2,413.1[b]
n200w140.005	**822**	76.6	819.6	99.7	515.2	90	521.0
Avg				99.49	338.2		399.8
Solved							24

Note. Optimal values in bold represent instances solved for the first time by BMR.*ng*.
[a]Best-known upper bound found by López-Ibáñez and Blum (2010).
[b]BMR.*ng* ran out of memory after 2,413.1 seconds. This time is not included in the final average CPU time.

Table 8 Comparison of *ng*-Tour and *ngL*-Tour Relaxations

Class	No. of instances	BMR.*ng* with *ng*-tour				BMR.*ng* with *ngL*-tour			
		Solved	%*LB*	T_{LB}	T_{TOT}	Solved	%*LB*	T_{LB}	T_{TOT}
Ascheuer easy	32	32	99.00	1.4	1.7	32	99.99	2.6	2.8
Ascheuer hard	18	18	99.60	23.5	33.4	18	99.70	89.2	114.1
Pesant	27	27	99.95	3.1	3.3	27	99.95	4.2	4.3
Potvin	28	28	99.50	8.9	9.7	28	99.67	11.7	12.2
Gendreau	140	140	99.59	19.1	37.7	140	99.78	35.6	36.9
Ohlmann	25	20	99.25	197.1	242.6	24	99.55	340.1	404.7
Avg			99.52	28.0	42.1		99.78	56.8	65.0
Sum	270	265				269			

Table 7 shows that all but one instance can be solved in a reasonable amount of computing time. Instance n200w140.004 could not be solved because BMR.*ng* ran out of memory (i.e., $|\mathcal{S}| > 10^8$). BMR.*ng* ran out of memory after 2,413.1 seconds at iteration $h = 7$, with $z^7_{ub} = 814$. Thus, $z^6_{ub} = 813$ is a valid lower bound on the optimal solution cost.

6.7. More Details on the Computational Results

Table 8 shows average results of the exact method, BMR.*ng*, using either *ng*-tour or *ngL*-tour relaxations on the six classes of instances. Columns "%*LB*" show that *ngL*-tour relaxation closes half of the gap left by *ng*-tour relaxation but is more time consuming. Columns "Solved" show the effectiveness of *ngL*-tour relaxation in solving the Ohlmann instances. BMR.*ng* using *ng*-tour relaxation cannot solve four of the instances (n150w120.002, n150w140.001, n200w120.005, and n200w140.001) that were solved by using *ngL*-tour relaxation.

Table 9 shows the effectiveness of the dominance and fathoming rules applied in Step 2 of the exact method (see §2). For a selected set of difficult instances, the table reports the following columns: the number of states fathomed by Fathoming Rule 1 (n_1), the percentage of times Fathoming Rule 1 is successfully applied ($\%n_1$), the number of states dominated by Dominance Rule 1 (n_2), the percentage of times Dominance Rule 1 is successfully applied ($\%n_2$), the number of states fathomed by Fathoming Rule 2 (n_3), the percentage of times Fathoming Rule 2 is successfully applied ($\%n_3$), the number of undominated states (n_4), and the percentage of undominated states on the number of states generated ($\%n_4$).

From Table 9, the effectiveness of the dominance and fathoming rules is noticeable. In particular, it is remarkable that 70%–90% of the states that are not discarded by Fathoming Rule 1 and Dominance Rule 1 can be fathomed with Fathoming Rule 2.

7. Conclusions

In this paper, we described an exact dynamic programming algorithm for the traveling salesman problem with time windows based on two tour relaxations proposed in the literature and on a new tour relaxation called *ngL*-tour.

We reported extensive computational results on several classes of both real-world and random instances taken from the literature and used them to test both exact and heuristic methods involving up to 233 vertices.

Table 9 Effectiveness of the Dominance and Fathoming Rules

Instance	Fath 1		Dom 1		Fath 2		Undominated	
	n_1	$\%n_1$	n_2	$\%n_2$	n_3	$\%n_3$	n_4	$\%n_4$
rbg049a	71	54.7	4	5.4	50	89.4	6	4.5
rbg050b	13	40.2	1	2.5	18	92.9	2	4.1
rbg050c	1,275	52.5	80	6.9	979	91.3	93	3.8
rbg193.2	6,394	70.6	1,027	38.6	1,219	74.6	414	4.6
rbg233.2	166,371	75.0	24,763	44.7	21,671	70.7	8,969	4.0
n150w120.002	369,823	76.5	6,281	5.5	95,468	88.7	12,149	2.5
n150w140.001	579,249	79.9	3,506	2.4	126,949	89.4	14,993	2.1
n200w120.002	629,067	77.7	5,461	3.0	154,832	88.4	20,223	2.5
n200w120.003	2,338,415	80.5	23,374	4.1	486,025	89.4	57,625	2.0
n200w120.005	1,389,485	81.8	9,951	3.2	270,429	90.3	29,072	1.7
n200w140.001	320,984	82.8	1,673	2.5	58,525	90.3	6,261	1.6

Note. $\%n_1 = n_1/(n_1 + n_2 + n_3 + n_4) \times 100$, $\%n_2 = n_2/(n_2 + n_3 + n_4) \times 100$, $\%n_3 = n_3/(n_3 + n_4) \times 100$, and $\%n_4 = n_4/(n_1 + n_2 + n_3 + n_4) \times 100$.

The exact algorithm solved all but one instance and outperforms all previously published exact methods for the TSPTW, solving 136 out of 270 instances for the first time.

Acknowledgment

The authors thank two anonymous referees for helpful comments.

References

Ascheuer, N., M. Fischetti, M. Grötschel. 2000. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks* **36**(2) 69–79.

Ascheuer, N., M. Fischetti, M. Grötschel. 2001. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Math. Programming Ser. A* **90**(3) 475–506.

Baker, E. K. 1983. An exact algorithm for the time-constrained traveling salesman problem. *Oper. Res.* **31**(5) 938–945.

Balas, E., N. Simonetti. 2001. Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS J. Comput.* **13**(1) 56–75.

Baldacci, R., A. Mingozzi, R. Roberti. 2011. New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.* Forthcoming.

Baldacci, R., E. Bartolini, A. Mingozzi, R. Roberti. 2010. An exact solution framework for a broad class of vehicle routing problems. *Comput. Management Sci.* **7**(3) 229–268.

Christofides, N., A. Mingozzi, P. Toth. 1981a. An algorithm for the time constrained travelling salesman problem. Technical Report IC OR 8125, Imperial College, London.

Christofides, N., A. Mingozzi, P. Toth. 1981b. State-space relaxation procedures for the computation of bounds to routing problems. *Networks* **11**(2) 145–164.

Dash, S., O. Günlük, A. Lodi, A. Tramontani. 2012. A time bucket formulation for the traveling salesman problem with time windows. *INFORMS J. Comput.* **24**(1) 132–147.

Desaulniers, G., F. Lessard, A. Hadjar. 2008. Tabu search, partial elementarity, and generalized *k*-path inequalities for the vehicle routing problem with time windows. *Transportation Sci.* **42**(3) 387–404.

Dumas, Y., J. Desrosiers, E. Gélinas, M. M. Solomon. 1995. An optimal algorithm for the traveling salesman problem with time windows. *Oper. Res.* **43**(2) 367–371.

Focacci, F., A. Lodi, M. Milano. 2002. A hybrid exact algorithm for the TSPTW. *INFORMS J. Comput.* **14**(4) 403–417.

Gendreau, M., A. Hertz, G. Laporte, M. Stan. 1998. A generalized insertion heuristic for the traveling salesman problem with time windows. *Oper. Res.* **46**(3) 330–335.

Jepsen, M., B. Petersen, S. Spoorendonk, D. Pisinger. 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* **56**(2) 497–511.

Langevin, A., M. Desrochers, J. Desrosiers, S. Gélinas, F. Soumis. 1993. A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks* **23**(7) 631–640.

Li, J. Q. 2009. A computational study of bi-directional dynamic programming for the traveling salesman problem with time windows. Working paper, University of California, Berkeley, Berkeley.

López-Ibáñez, M., C. Blum. 2010. Beam-ACO for the travelling salesman problem with time windows. *Comput. Oper. Res.* **37**(9) 1570–1583.

Mingozzi, A., L. Bianco, S. Ricciardelli. 1997. Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Oper. Res.* **45**(3) 365–377.

Ohlmann, J. W., B. W. Thomas. 2007. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS J. Comput.* **19**(1) 80–90.

Ohlmann, J., B. Thomas. 2010. Private communication. (April 7).

Pesant, G., M. Gendreau, J.-Y. Potvin, J.-M. Rousseau. 1998. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. *Transportation Sci.* **32**(1) 12–29.

Potvin, J.-Y., S. Bengio. 1996. The vehicle routing problem with time windows part II: Genetic search. *INFORMS J. Comput.* **8**(2) 165–172.

Righini, G., M. Salani. 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optim.* **3**(3) 255–273.

Wolfler Calvo, R. 2000. A new heuristic for the traveling salesman problem with time windows. *Transportation Sci.* **34**(1) 113–124.