Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

An Optimal Algorithm for the Traveling Salesman Problem with Time Windows

Yvan Dumas, Jacques Desrosiers, Eric Gelinas, Marius M. Solomon,

To cite this article:
Yvan Dumas, Jacques Desrosiers, Eric Gelinas, Marius M. Solomon, (1995) An Optimal Algorithm for the Traveling Salesman Problem with Time Windows. Operations Research 43(2):367-371. https://doi.org/10.1287/opre.43.2.367

TECHNICAL NOTE

AN OPTIMAL ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM WITH TIME WINDOWS

YVAN DUMAS, JACQUES DESROSIERS and ERIC GELINAS

GERAD and HEC, Montreal, Canada

MARIUS M. SOLOMON

Northeastern University, Boston, Massachusetts

(Received October 1991; revisions received November 1992, April 1993; accepted July 1994)

This paper presents the development of new elimination tests which greatly enhance the performance of a relatively well established dynamic programming approach and its application to the minimization of the total traveling cost for the traveling salesman problem with time windows. The tests take advantage of the time window constraints to significantly reduce the state space and the number of state transitions. These reductions are performed both a priori and during the execution of the algorithm. The approach does not experience problems stemming from increasing problem size, wider or overlapping time windows, or an increasing number of states nearly as rapidly as other methods. Our computational results indicate that the algorithm was successful in solving problems with up to 200 nodes and fairly wide time windows. When the density of the nodes in the geographical region was kept constant as the problem size was increased, the algorithm was capable of solving problems with up to 800 nodes. For these problems, the CPU time increased linearly with problem size. These problem sizes are much larger than those of problems previously reported in the literature.

The traveling salesman problem (TSP) with time windows (**TSPTW**) involves the design of a minimum cost path for a vehicle which visits a set of nodes. Each node is visited exactly once and the service at a node must begin within the time window defined by the earliest and the latest time when the start of service is permitted at that node. If a vehicle arrives at a node too early, it will wait. In addition, due dates cannot be violated.

Time windows arise naturally in problems faced by business organizations which work on fixed time schedules. The **TSPTW** has important practical applications in bank or postal deliveries, school-bus routing and scheduling, and automated manufacturing environments. It also constitutes a key component of more complex vehicle routing problems; in particular, it is an essential module of "cluster-first, route second" approaches to such problems.

While the research on time constrained routing problems has grown at an explosive rate, research on the **TSPTW** has been scant. Savelsbergh (1985) has shown that even finding a feasible solution to the **TSPTW** is an NP-complete problem and has, therefore, proposed interchange heuristics. Nevertheless, several other authors have focused on exact algorithms to minimize the total schedule time. Christofides, Mingozzi and Toth (1981) propose a branch-and-bound algorithm where lower bounds are derived from state-space relaxations of dynamic programs. The approach solved 50-node problems with moderately tight time windows. Baker (1983) also presents a branch-and-bound approach where lower bounds are obtained from the dual of a relaxation of the proposed model. The algorithm performed well on problems with up to 50 nodes when only a small percentage of the time windows overlap. Langevin et al. (1990) present a linear multicommodity flow formulation involving two complementary flows. It can handle either the total traveling time or the total schedule time objective. The authors report the solution of problems with up to 40 nodes.

Related research has also been performed on the TSP with precedence constraints, on special routing structures and on alternative objective functions. This work and many other time constrained routing problems are surveyed by Desrosiers, Solomon and Soumis (1992).

Even though the **TSPTW** is a special case of the vehicle routing problem with time windows, the best known approach to the latter problem (Desrochers, Desrosiers and Solomon 1992) is not well suited to solve the **TSPTW**. This column generation approach would experience extreme degeneracy difficulties in this case.

The contribution of this paper is the development of new elimination tests which greatly enhance the performance of a relatively well established dynamic programming approach and its application to the minimization of the total traveling cost for the **TSPTW**. The tests take advantage of the time window constraints to significantly

Subject classifications: Dynamic programming/optimal control: applications. Transportation: vehicle routing.
Area of review: DISTRIBUTION, TRANSPORTATION AND LOGISTICS.

0030-364X/95/4302-0367 $01.25

decrease the state space and the number of state transitions. These reductions are performed both a priori and during the execution of the algorithm. This approach is capable of optimally solving realistic test problems with fairly wide, overlapping time windows and of much larger size than previously reported in the literature. Furthermore, for certain classes of problems, its computational behavior is linear in problem size.

Our method does not experience problems stemming from increasing problem size, wider, or overlapping time windows, nearly as rapidly as the method of Baker. Furthermore, it can be used as a standalone method to optimally solve large size problems. Other dynamic programming approaches, such as those of Christofides, Mingozzi and Toth, and Baker, use relaxed recursions only to obtain lower bounds which are then embedded in a branch-and-bound scheme, for an easier objective function than that treated in this paper. Finally, it does not experience numerical instability difficulties as does the procedure of Langevin et al.

1. A DYNAMIC PROGRAMMING FORMULATION

Consider a network $G = (N, A)$ where $N = \{1, \ldots, n\}$ is the set of nodes and A is the set of feasible arcs. Associate with each node $i \in N$ a time window $[a_i, b_i]$ and a service time s_i. Associate with each arc a duration t_{ij} and a cost c_{ij}. An arc $(i, j) \in A$ is feasible if $a_i + s_i + t_{ij} \leq b_j$. Let N' be defined as $N - \{n\}$. A path in the network G is defined as a sequence of nodes $i_1, i_2, \ldots, i_k$ such that each arc (i_j, i_{j+1}) belongs to A and the time service begins at node i_j, $j = 1, \ldots, k$, is within the time window of that node. Let the time of departure from node 1 be a_1 and the time service begins at node i be t_i, $i \in N$. If a path goes from nodes i to j, the time service begins at node j is given by $t_j = \max\{t_i + s_i + t_{ij}, a_j\}$ if $t_i + s_i + t_{ij} \leq b_j$, and $t_j = \infty$, otherwise. This definition implies waiting at node j if the path arrives there too early. For the optimal **TSPTW** solution, we require a least-cost feasible path starting at node 1 and ending at node n that visits every node in G exactly once.

Define now $F(S, i, t)$ as the least cost of a path starting at node 1, passing through every node of $S \subseteq N'$ exactly once, ending at node $i \in S$, and ready to service node i at time t or later. Since the cost $F(S, i, t)$ and the time t need to be addressed separately, a two-dimensional labeling is necessary. This also makes the dynamic programming minimization of the total traveling time more difficult than that of the total schedule time. The function $F(S, i, t)$ can be computed by solving the recurrence equations:

$$F(S, j, t) = \min_{(i,j)\in A} \{F(S - \{j\}, i, t') + c_{ij} | t \geq t' + s_i + t_{ij}, a_j \leq t' \leq b_j\} \quad (1)$$

for all $S \subseteq N'$, $j \in S$ and $a_j \leq t \leq b_j$. The recursion formula is initialized by

$$F(\{1, j\}, j, t) = c_{1j} \quad \text{if } (1, j) \in A, \text{ and } F(\{1, j\}, j, t) = \infty \quad \text{otherwise,} \quad (2)$$

where $a_j \leq t \leq b_j$, $t = \max\{a_1 + s_1 + t_{1j}, a_j\}$. The optimal **TSPTW** solution is given by:

$$\min_{(i,n)\in A} \min_{a_i \leq t \leq b_i} \{F(N, i, t) + c_{in} | t \geq b_n - t_{in} - s_i\}. \quad (3)$$

Formulas (1)–(3) define a shortest path algorithm on a state graph whose nodes are the states (S, i, t) and whose arcs represent transitions from one state to another. This algorithm is a forward dynamic programming algorithm where at step s, $s = 1, \ldots, n - 1$, a path of length s is generated. The states (S, i, t) of cost $F(S, i, t)$ are defined as: S is an unordered set of visited nodes, i is the last visited node, $i \in S$, and t is the time service begins at node i. Note that there are several paths that visit set S and end at i. Among them, we only conserve the Pareto optimal elements. In other words, given (S, i, t^1) and (S, i, t^2), the second state can be eliminated if $t^1 \leq t^2$ and $F(S, i, t^1) \leq F(S, i, t^2)$. A number of applications of dominance between states as a function of time and cost values can be found in the literature (see, for example, Kolen, Rinnooy Kan and Trienekens 1987, Desrochers and Soumis 1988).

The number of states (S, i, t), $a_i \leq t \leq b_i$ is countable in t if we use integer data to define the time windows and the traveling times. In addition, the function $F(S, i, t)$ is stepwise decreasing as a function of t over the interval $[a_i, b_i]$. After the dominance test, the two-dimensional labels $(t, F(S, i, t))$ of (S, i) can be ordered by increasing time and decreasing cost value. Let $FIRST(S, i)$ be the time value of the first label of that list.

2. POST FEASIBILITY TESTS

For the TSP, the most important and difficult constraint is to visit all nodes. When time windows or precedence constraints are present, these impose a partial ordering of the nodes. Great computational efficiency can be derived from eliminating partial paths that do not satisfy such orderings. The tests presented here detect when a node cannot extend a current partial path, thereby permitting the elimination of such paths.

To describe them, let $EAT(i, j)$ be the earliest arrival time at node j from node i. This can be computed by solving a shortest time path problem, where the path starts at time a_i and satisfies the time window constraints from i to j. Also let $LDT(i, j)$ be the latest departure time from i such that t_j is feasible. Likewise, this can be computed by solving a constrained shortest time path problem, where the path starts at node j at time b_j and uses the reverse arc directions. Now define $BEFORE(j)$ to be the set of all nodes which must necessarily be visited before j. Hence, $BEFORE(j) = \{k \in N | EAT(k, j) > b_j\}$. Note that the values $EAT(i, j)$ and

Published in Operations Research on April 01, 1995 as DOI: 10.1287/opre.43.2.367.
This article has not been copyedited or formatted. The final version may differ from this version.

$LDT(i, j)$ can be overestimated or underestimated, respectively, by using an unconstrained time path. This is much faster to compute and it gives satisfactory results in practice. Furthermore, if travel times satisfy the triangle inequality, no shortest time paths need to be computed as, for example, $EAT(i, j) > b_j$ can be replaced by $a_i + s_i + t_{ij} > b_j$. Finally, we say that a state (S, i, t) admits a feasible extension toward j, i.e., the state $(S \cup \{j\}, j, max\{a_j, t + s_i + t_{ij}\})$ can be created if $t + s_i + t_{ij} \leq b_j$.

The post-feasibility tests that we present next are similar to the ones in Desrosiers, Dumas and Soumis (1986) developed for the single vehicle pick-up and delivery problem with time windows. An important difference, however, is that while this problem necessitates a more complex formulation, it is easier to solve by dynamic programming than is the **TSPTW**. This is because the precedence constraints between the pick-up and the delivery location associated with the same request help eliminate a significant number of states and state transitions. To obtain significant reductions for the **TSPTW**, we had to create a precedence structure as defined by $BEFORE(j)$, for each node j. This will be used in test 2 below.

The first test involves the comparison between two values, $FIRST(S, i)$ and $\min_{j \notin S} LDT(i, j)$ and it has an effect on all the successors of i.

Test 1. Given the states (S, i, t) for all $a_i \leq t \leq b_i$ if the smallest time value to begin service at node i is greater than the latest feasible departure time toward j, for all j, i.e.,

$$FIRST(S, i) > \min_{j \notin S} LDT(i, j),$$

then the states (S, i, t) for all $a_i \leq t \leq b_i$ do not admit feasible extensions toward any node.

This global test eliminates the states (S, i, t) for all t by only examining the earliest time to begin service at i. If for this time a feasible arrival at some other node is impossible, then all the states (S, i, t) are eliminated. In this case, the states to be treated next are those with a new ending node i and the same set S; when all ending nodes i have been considered, a new set S is examined.

The next test consists of examining whether one set S is included in another, $BEFORE(j)$. Its effect is only on one successor of i.

Test 2. Given the states (S, i, t) for all $a_i \leq t \leq b_i$ and given node j, $j \notin S$, $(i, j) \in A$, if $BEFORE(j) \not\subset S$, then no feasible extensions exist toward j.

This test states that all "predecessors" of node j must necessarily be visited before visiting j. If the test succeeds in eliminating node j as a successor for i, the next step is to test another potential successor, or to apply test 1 to a new set of states with a different i.

The final test examines the future of a state. Its effect is local: It only concerns the extension of a single label of (S, i).

Test 3. Given the state (S, i, t) for a fixed t, $a_i \leq t \leq b_i$ and given node j, $j \notin S$, $(i, j) \in A$, if (S, i, t) can be extended to j, i.e., $t \leq LDT(i, j)$, but cannot be extended further to some k, $k \notin S$, $(i, k) \in A$, i.e., $t + t_{ij} > LDT(j, k)$, then j cannot succeed i for (S, i, t) and the states (S, i, t'), $t' \geq t$ are not extended toward j.

If the new label $(S \cup \{j\}, j, \max\{a_j, t + s_i + t_{ij}\})$ is not created, a new node j is considered, because no other labels can be feasible for time values greater than t.

3. COMPUTATIONAL EXPERIMENTS

The algorithm was coded in C and the experiments were conducted on a Hewlett-Packard workstation HP9000/730. During preprocessing, the arc set A was reduced to A' by applying **TSPTW** specific rules (Langevin et al.). The time windows were also reduced by applying the rules described in Desrochers, Desrosiers and Solomon.

The algorithm was tested on two sets of problems. The first set consists of symmetric Euclidean problems described in Langevin et al., where customer coordinates are uniformly distributed between 0 and 50 and travel times equal distances. The time windows are generated around the times to begin service at each customer of a second nearest neighbor TSP tour. Each side of a time window is generated as a uniform random variable in the interval $[0, w/2]$, where $w = 20$, 40, 60, 80, and 100. The problems exhibit various degrees of difficulty. First, the second nearest neighbor tour is not near optimal for the **TSPTW**. Experiments conducted for $n = 60$ and all w indicate that the number of common arcs in the two tours is only about 20%. Second, for a given problem size, problem difficulty increases with the time windows' width, because the percentage of overlapping time windows increases. Third, it also increases significantly with problem size because the geographical area remains constant. Hence, as n increases, the density of the points in this area increases. In turn, this decreases the ability of the time windows to reduce the number of feasible tours.

The results are presented in Table I. They indicate that our algorithm was successful in solving problems with up to 200 nodes and fairly wide time windows. As expected, the CPU time increases with time window width and with problem size. Using the logarithm in base 10 of the CPU time, it can be seen that for a given problem size the behavior of the algorithm is exponential with respect to time window width. Nevertheless, for narrow widths, its behavior is less than exponential allowing larger size problems to be solved. For example, a 250-node problem with $w = 20$ is easy to solve in less than 10 seconds.

Note that the algorithm is designed to handle functions capturing the additional cost for waiting at a node, such as the total schedule time. In this case, by letting $F(S, i,$

Table I
Computational Results for the **TSPTW**

n	w	$\|A'\|$	Init	Opt	$\|S\|$	$\|(S, i)\|$	$\|(S, i, t)\|$	CPU	Log
20	20	34.2	375.0	361.2	2.8	4.4	4.4	0.02	−1.63
20	40	72.2	349.8	316.0	7.0	14.8	19.6	0.05	−1.33
20	60	115.2	374.4	309.8	14.8	33.8	35.6	0.14	−0.85
20	80	143.8	373.6	311.0	22.4	54.0	64.0	0.23	−0.63
20	100	207.0	373.0	275.2	86.2	244.8	291.4	1.27	0.10
40	20	121.8	526.4	486.6	5.4	9.6	16.0	0.08	−1.12
40	40	230.2	521.0	461.0	14.0	33.0	55.0	0.24	−0.61
40	60	404.2	507.6	416.4	154.6	428.2	554.0	4.37	0.64
40	80	488.8	508.4	399.8	268.2	740.6	890.4	7.52	0.88
40	100	640.8	486.6	377.0	636.6	2644.4	5298.4	31.40	1.50
60	20	226.0	632.8	581.6	7.8	15.2	21.8	0.15	−0.81
60	40	462.8	662.6	590.2	38.6	103.2	145.8	0.89	−0.05
60	60	670.6	684.8	560.0	243.6	626.8	1344.4	6.84	0.83
60	80	987.4	652.6	508.0	1151.8	3526.0	8414.2	46.62	1.67
60	100	1219.8	649.6	514.8	2897.8	8465.8	20846.4	199.84	2.30
80	20	362.6	753.8	676.6	11.0	28.2	49.8	0.35	−0.46
80	40	755.6	730.4	630.0	77.0	198.0	359.0	2.68	0.43
80	60	1149.6	732.6	606.4	843.8	3232.8	7716.8	55.32	1.74
80	80	1490.8	735.8	593.8	3179.4	10449.2	16419.0	220.29	2.34
100	20	510.0	826.4	757.6	14.6	33.4	45.0	0.62	−0.21
100	40	1051.2	830.6	701.8	143.2	442.6	1059.4	7.40	0.87
100	60	1731.4	828.4	696.6	1589.6	4432.2	6804.8	107.95	2.03
150	20	975.8	982.2	868.4	47.4	137.0	326.8	2.44	0.39
150	40	2128.0	996.4	834.8	1484.8	5857.4	20371.0	115.86	2.06
150	60	2953.6	996.6	805.0	4144.6	13158.0	26351.0	462.97	2.67
200	20	1655.6	1143.6	1009.0	60.8	195.8	480.4	6.65	0.82
200	40	3351.2	1163.8	984.2	2380.6	8391.4	14240.0	251.41	2.40

n: the number of nodes;
w: the width of the time windows;
$|A'|$: the arc size of the reduced network;
Init: the average over five problems of the initial solutions obtained by a second nearest neighbor heuristic for the TSP;
Opt: the average over five problems of the optimal solutions for the **TSPTW** (for $n = 100$ and $w = 60$, the average is only over three problems as the memory limitation of 50 Mb was exceeded for the other two problems);
$|S|$, $|(S, i)|$ and $|(S, i, t)|$: the maximum number of labels created;
CPU: the CPU time in seconds on a Hewlett-Packard workstation (HP9000/730, 76 mips, 22 M flops);
log: the logarithm in base 10 of the CPU time.

$t) = t - a_1$ and preserving only the state with the smallest t, the states (S, i, t) can be reduced to (S, i). Comparing the results depicted in columns seven and eight of Table I, it is apparent that the minimization of the total schedule time is an easier dynamic programming optimization problem than the one treated in this paper.

We also analyzed the separate impact of each test. As can be seen from Table II, test 2 is the most beneficial. Without it we were not able to solve a problem of much smaller size than the ones solved when we utilized this test. Test 1 is the least powerful because much of its work was performed by reducing the width of the time windows in the preprocessing phase.

We also tested the algorithm on problems where the density of the points remains constant. We generated them by starting from the geographical and time window characteristics of the problems described above and doubling the geographical area each time the number of points doubles. In these experiments, we fixed w at 60, to provide for fairly wide time windows. Note that the number of common arcs between the second nearest neighbor TSP tour and the optimal **TSPTW** tour stayed at about 20% even for the 800-node problems. Figure 1 illustrates that the computational behavior of the algorithm is linear in the number of nodes. Hence, we were capable of solving problems with up to 800 nodes in about 650 seconds. For larger problems, the method did not experience CPU time difficulties. However, it encountered memory problems.

4. CONCLUSIONS AND EXTENSIONS

This paper presented the development of a very effective optimal dynamic programming algorithm for the minimization of the total traveling cost for the **TSPTW**. Its effectiveness stems from the post-feasibility tests which

Published in Operations Research on April 01, 1995 as DOI: 10.1287/opre.43.2.367.
This article has not been copyedited or formatted. The final version may differ from this version.

TABLE II
The Impact of Each Test on the Behavior of the Algorithm ($n = 60$)

w =		All Tests	Test 1 Removed	Test 2 Removed	Test 3 Removed
20	$\|(S, i, t)\|$	21.8	21.8	3716.4	37.4
	CPU	0.15	0.15	6.20	0.17
40	$\|(S, i, t)\|$	145.8	145.8	18520.5	229.0
	CPU	0.91	0.91	49.23	0.98
60	$\|(S, i, t)\|$	1344.4	1346.2	*	3864.8
	CPU	6.84	7.42	*	9.54

exploit the time window constraints to substantially reduce the state space and the number of state transitions. These eliminations are derived from trying to validate the most important **TSPTW** constraint which imposes the visit of all the nodes. Consequently, our algorithm starts experiencing memory and CPU time difficulties only for much larger problem sizes and more difficult time window structures than previously reported in the literature.

Several important extensions can be seen. First, as can be observed from test 2, the algorithm can handle additional precedence constraints. Second, because the method is a forward dynamic program, it can optimize more general cost functions, such as step functions, and nonlinear functions. As long as the cost function is a nondecreasing function of time, the algorithm and the dominance process can be adapted. Furthermore, when there are several nodes at the same physical location, but with different time windows, the algorithm can be accelerated by removing a priori a set of arcs between these nodes. This is the "same location criterion" used in Dumas, Desrosiers and Soumis (1991).

The optimal algorithm presented in this paper can be transformed into a much faster heuristic by removing a priori some unattractive arcs or by controlling and limiting the number of states created. This type of relaxation can provide a good solution to much larger or very difficult problems.

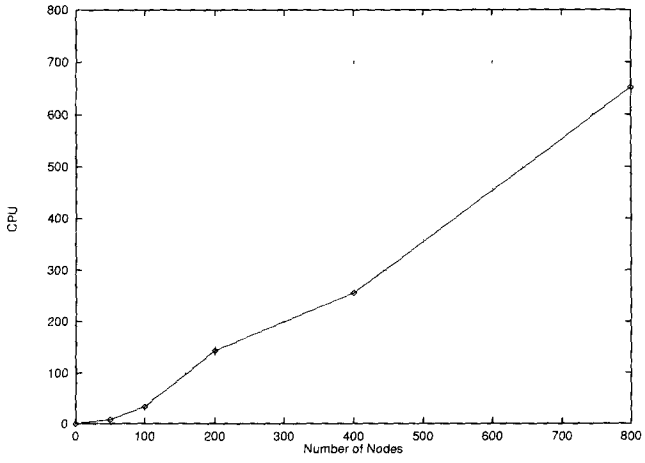

Figure 1. Computational behavior of the algorithm on constant point density problems.

The method can also be used as a sensitivity analysis tool for the TSP. Given a heuristic solution for the TSP, one can impose cost windows around the cost of the solution at each node. The width of each cost window equals twice the travel cost to each node's nearest, second nearest, and third nearest neighbor, respectively. Our algorithm can then be used to explore the neighborhood of solutions.

ACKNOWLEDGMENT

The authors benefited from fruitful discussions with François Soumis and Sylvie Gelinas. The authors also wish to thank Pierre Girard for his programming help. The research of the fourth author was partially supported by the Research Centers GERAD and CRT, Montreal, Canada and by the Patrick F. and Helen C. Walsh Research Professorship.

REFERENCES

BAKER, E. 1983. An Exact Algorithm for the Time Constrained Traveling Salesman Problem. *Opns. Res.* **31,** 938–945.

CHRISTOFIDES, N., A. MINGOZZI AND P. TOTH. 1981. State Space Relaxation Procedures for the Computation of Bounds to Routing Problems. *Networks* **11,** 145–164.

DESROCHERS, M., AND F. SOUMIS. 1988. A Reoptimization Algorithm for the Shortest Path Problem With Time Windows. *Eur. J. Opnl. Res.* **35,** 242–254.

DESROCHERS, M., J. DESROSIERS AND M. SOLOMON. 1992. A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows. *Opns. Res.* **40,** 342–354.

DESROSIERS, J., Y. DUMAS AND F. SOUMIS. 1986. A Dynamic Programming Solution of the Large-Scale Single-Vehicle Dial-a-Ride Problem With Time Windows. *Am. J. Math. and Mgmt. Sci.* **6,** 301–325.

DESROSIERS, J., M. SOLOMON AND F. SOUMIS. 1992. Time Constrained Routing and Scheduling. In *Handbooks in Operations Research/Management Science*, Volume on *Networks*, M. Ball, T. Magnanti, C. Monma and G. Nemhauser (eds.). Elsevier Science, Amsterdam, The Netherlands (to appear).

DUMAS, Y., J. DESROSIERS AND F. SOUMIS. 1991. The Pick-up and Delivery Problem With Time Windows. *Eur. J. Opnl. Res.* **54,** 7–22.

KOLEN, A., A. RINNOOY KAN AND H. TRIENEKENS. 1987. Vehicle Routing With Time Windows. *Opns. Res.* **35,** 266–273.

LANGEVIN, A., M. DESROCHERS, J. DESROSIERS AND F. SOUMIS. 1990. A Two-Commodity Flow Formulation for the Traveling Salesman and the Makespan Problems With Time Windows. Working Paper CRT-732, Centre de Recherche sur les Transports, Montreal, Canada.

SAVELSBERGH, M. 1985. Local Search in Routing Problems With Time Windows. *Anns. Opns. Res.* **4,** 285–305.