Implementation of a Linear Time Algorithm for Certain Generalized Traveling Salesman Problems

Neil Simonetti and Egon Balas*

Carnegie Mellon University, Pittsburgh PA 15213, USA

Abstract. This paper discusses an implementation of a dynamic programming approach to the traveling salesman problem that runs in time linear in the number of cities. Optimality can be guaranteed when precedence constraints of a certain type are present, and many problems involving time windows fall into this class. Perhaps the most interesting feature of the procedure is that an auxiliary structure is built before any particular problem instance is known, reducing the computational effort required to solve a given problem instance to a fraction of what it would be without such a structure.

1 Introduction

This paper discusses an implementation of the approach proposed in [2] for solving some classes of generalized traveling salesman problems (TSP), symmetric or asymmetric, by finding a shortest path in an auxiliary digraph constructed for this purpose. The algorithm runs in time linear in the number of cities, and has a variety of uses, from solving instances of the TSP with time windows, to serving to improve a given heuristic solution to a standard TSP. At this point, the algorithm has been used on TSP problems with time windows, found in the literature [1, 3, 6, 9], has improved some standard TSP solutions generated by the Kanellakis & Papadimitriou heuristic [4], and has solved a new class of problems we call the TSP with time targets.

The main novel feature of this implementation is that an auxiliary structure is built in advance, *without prior knowledge of the problem instance*. This structure then serves as a foundation on which the algorithm runs, vastly reducing the computational effort required for solving any particular problem instance.

Section 2 gives a quick overview of the problem. Section 3 outlines how the auxiliary structure is built and used. Section 4 shows current computational results.

2 Background

The approach proposed in [2] starts from a precedence-constrained n-city TSP, symmetric or asymetric, defined on a complete graph G, directed or undirected,

* Research supported by Grant DMI-9201340 of the National Science Foundation and contract N00014-89-J-1063 of the Office of Naval Research.

with city 1 fixed as the home city where all tours must start and finish. The precedence constraints are given by:

Problem 1:

(i) a positive integer $k < n$.
(ii) an ordering $\{1, \ldots, n\}$ of the set N of cities, such that there exists an optimal permutation π of $\{1, \ldots, n\}$ (and associated tour) with the property that
(iii) for all $i, j \in N$, $j \geq i + k$ implies $\pi(i) < \pi(j)$.

The idea behind these precedence constraints is that if in the initial ordering city j comes k or more places after city i, then city j must be visited after city i in an optimal tour.

The method for solving the problem involves two steps: (1) building a special auxiliary digraph, G^*; and (2) solving a shortest path problem on G^*. G^* has $n+1$ *layers*, one layer for each position in the tour, with the home city appearing at both the beginning and end of the tour. The first and last layers of G^* each have only one node since only the home city can be at the beginning and end of the tour. If s is the name of the node in layer 1, and t is the name of the node in layer $n + 1$, then there is a 1-1 correspondence between feasible tours in G satisfying (i), (ii), and (iii) and $s - t$ paths in G^*. Furthermore, optimal tours correspond to shortest paths.

Every node in the ith layer of G^* corresponds to a unique state specifying which city is in position i, and which cities are visited in positions 1 through $i - 1$. This state can be expressed by the three elements:

1. j, the city in position i.
2. S^-, the set of cities numbered i or higher that are visited in one of the positions 1 through $i - 1$. ($S^- := \{h \in N : h \geq i,\ \pi(h) < i\}$)
3. S^+, the set of cities numbered below i that are not visited in one of the positions 1 through $i - 1$. ($S^+ := \{h \in N : h < i,\ \pi(h) \geq i\}$) (Note that this implies $|S^-| = |S^+|$.)

Nodes in G^* can be referenced by the notation (i, j, S^-, S^+). When referencing a node in a certain (possibly arbitrary) layer i, the notation will simply be (j, S^-, S^+), where the elements are dependent on i. G^* contains roughly $n(k+1)2^{k-2}$ nodes. All the arcs of G^* connect nodes of adjacent layers. There is an arc from a node in layer i to one in layer $i+1$ if the states represented by the two nodes can be part of the same tour. When this occurs, the nodes are said to be *compatible*. The cost assigned to the arc connecting node (i, j, S^-, S^+) to node $(i + 1, l, T^-, T^+)$ is the cost of the arc (j, l) in the original TSP instance. No node of G^* has an in-degree greater than k, which bounds the number of arcs at $nk(k + 1)2^{k-2}$. For further details on G^*, see [2].

One generalization of the original problem is to allow different values of k for the precedence constraint (iii) for different cities i. The conditions (i) and (iii) change as follows:

Problem 2:

(i) a family of positive integers $\{k(i) : i \in N\}$ all less than n.
(iii) for all $i, j \in N$, $j \geq i + k(i)$ implies $\pi(i) < \pi(j)$.

The following section deals with issues of implementing the algorithm for problem 2.

3 The Auxiliary Structure

We will use the following conventions when dealing with the cities of the TSP and the nodes of auxiliary structures:

"Cities" refers to vertices in G, all found in the set N.
"Nodes" refers to vertices in auxiliary structures, such as G^*.
$min(\emptyset) := \infty$; $max(\emptyset) := -\infty$.

While the auxiliary digraph G^* is problem dependent, the building blocks of G^* have a structure that allows them to be constructed before the problem instance is known. This is very important because the construction of the digraph is much more difficult than finding its shortest path. We know of no other combinatorial algorithm in which most of the extensive calculations are worked out and saved before the problem is examined.

The nodes of G^* can be partitioned into $n+1$ layers, each layer corresponding to a position in the tour. For problem 1, all the layers, except the first and last k layers, consist of identical copies of a set W_k^* of $(k+1)2^{k-2}$ nodes, and the remaining layers are subsets of W_k^*. The compatibility test mentioned in Section 2 for determining arcs of G^* can also be applied to W_k^*. Furthermore, if we construct a layered digraph G_k^{**} by using singleton nodes for the first and last layers (call them s and t) and $n-1$ copies of W_k^* for the remaining layers, and use the compatibility test to determine the arcs of G_k^{**} (though not their costs); then the sets of $s-t$ paths in G^* and G_k^{**} are identical. The nodes of W_k^* will be referenced using the notation (j, S^-, S^+), where the values of j, S^-, and S^+ (defined in Section 2 above) depend only on the value of i, which indicates the specific layer of G_k^{**}.

W_k^* and the set of compatible pairs from W_k^* can be built without prior knowledge of the problem, except for the value k. But since $W_h^* \subset W_m^*$ whenever $h \leq m$, by building this structure (i.e. the node set W_m^* and its set of compatible pairs) for a single "large" value of m, we can solve any problem instance with $k \leq m$ once this structure is loaded into memory.

For problem 2, all the layers of G^* are subsets of W_K^*, where $K := max\{k(i) : i \in N\}$, and the set of arcs of G^* between two layers is a subset of the compatible pairs of W_K^*. Thus G^* is still a subgraph of G_K^{**}, and W_K^* and its compatible pairs can be built beforehand. However, there may be $s-t$ paths in G_K^{**} that are not in G^*. One way to avoid examining some $s-t$ paths in G_K^{**} that are not in G^* is to replace the copy of W_K^* for layer i with a copy of $W_{m_i}^*$ for some $m_i \leq K$ which will not remove any $s-t$ path also in G^*. Call this slimmer

	No. Node Label (i, j, S^-, S^+)	Compatible Predecessors	Compatible Successors
Level 1:	1: $(i, i, \emptyset, \emptyset)$	1,3,8,20	1,2,4,9
Level 2:	2: $(i, i+1, \emptyset, \emptyset)$	1,3,8,20	3,5,10
	3: $(i, i-1, \{i\}, \{i-1\})$	2,6,16	1,2,4,9
Level 3:	4: $(i, i+2, \emptyset, \emptyset)$	1,3,8,20	6,7,11
	5: $(i, i+1, \{i\}, \{i-1\})$	2,6,16	8,15
	6: $(i, i-1, \{i+1\}, \{i-1\})$	4,12	3,5,10
	7: $(i, i, \{i+1\}, \{i-1\})$	4,12	8,15
	8: $(i, i-2, \{i\}, \{i-2\})$	5,7,19	1,2,4,9
Level 4:	9: $(i, i+3, \emptyset, \emptyset)$	1,3,8,20	12,13,14
	10: $(i, i+2, \{i\}, \{i-1\})$	2,6,16	16,17
	11: $(i, i+2, \{i+1\}, \{i-1\})$	4,12	18,19
	12: $(i, i-1, \{i+2\}, \{i-1\})$	9	6,7,11
	13: $(i, i, \{i+2\}, \{i-1\})$	9	16,17
	14: $(i, i+1, \{i+2\}, \{i-1\})$	9	18,19
	15: $(i, i+1, \{i\}, \{i-2\})$	5,7,19	20
	16: $(i, i-2, \{i+1\}, \{i-2\})$	10,13	3,5,10
	17: $(i, i, \{i+1\}, \{i-2\})$	10,13	20
	18: $(i, i-1, \{i, i+1\}, \{i-2, i-1\})$	11,14	20
	19: $(i, i-2, \{i, i+1\}, \{i-2, i-1\})$	11,14	8,15
	20: $(i, i-3, \{i\}, \{i-3\})$	15,17,18	1,2,4,9

Fig. 1. Nodes of W_4^* and its compatible pairs.

auxiliary structure (which is dependant on the problem instance) G^{**}. Given the values m_i, the nodes in layer i of G^{**} are simply $W^*_{m_i}$. The arcs connecting two adjacent layers, i and $i+1$, of G^{**} are also easily found from the list of compatible pairs (u, v) of $W_K^* \times W_K^*$ by choosing the pairs (u, v) with $u \in W^*_{m_i}$ and $v \in W^*_{m_{i+1}}$. The rest of this section deals with calculating good values of m_i when choosing $W^*_{m_i}$ to represent a layer of G^{**}, and ways to avoid examining the remaining $s - t$ paths in G^{**} that are not also in G^*.

Since we have $W_h^* \subset W_m^*$ whenever $h \leq m$, there is a natural division of the nodes of W_m^* into *levels*. Let W_1^* be the first level of nodes in W_m^*, and then define the hth level of nodes $(2 \leq h \leq m)$ to be those in the set $W_h^* \setminus W_{h-1}^*$. Let $L(v)$ be the level in which node v belongs for a given node $v := (j, S^-, S^+) \in W_K^*$. Figure 1 illustrates the breakdown of W_4^* into levels. The set of compatible pairs is expressed as a list of successors or predecessors.

Figure 2 shows the graphs G^*, G^{**} and G_K^{**} for the case of problem 2 where $n = 9$, $k(3) = 4$, and $k(i) = 3$ for $i \neq 3$. The paths shown in the figure correspond to the feasible tour 1-4-2-6-3-5-7-9-8-1; the infeasible tour 1-2-3-4-5-8-7-9-6-1, which extends beyond G^{**}; and the infeasible tour 1-3-2-5-7-4-8-6-9-1, which does not extend beyond G^{**}, but must be avoided because it contains nodes not in G^*.

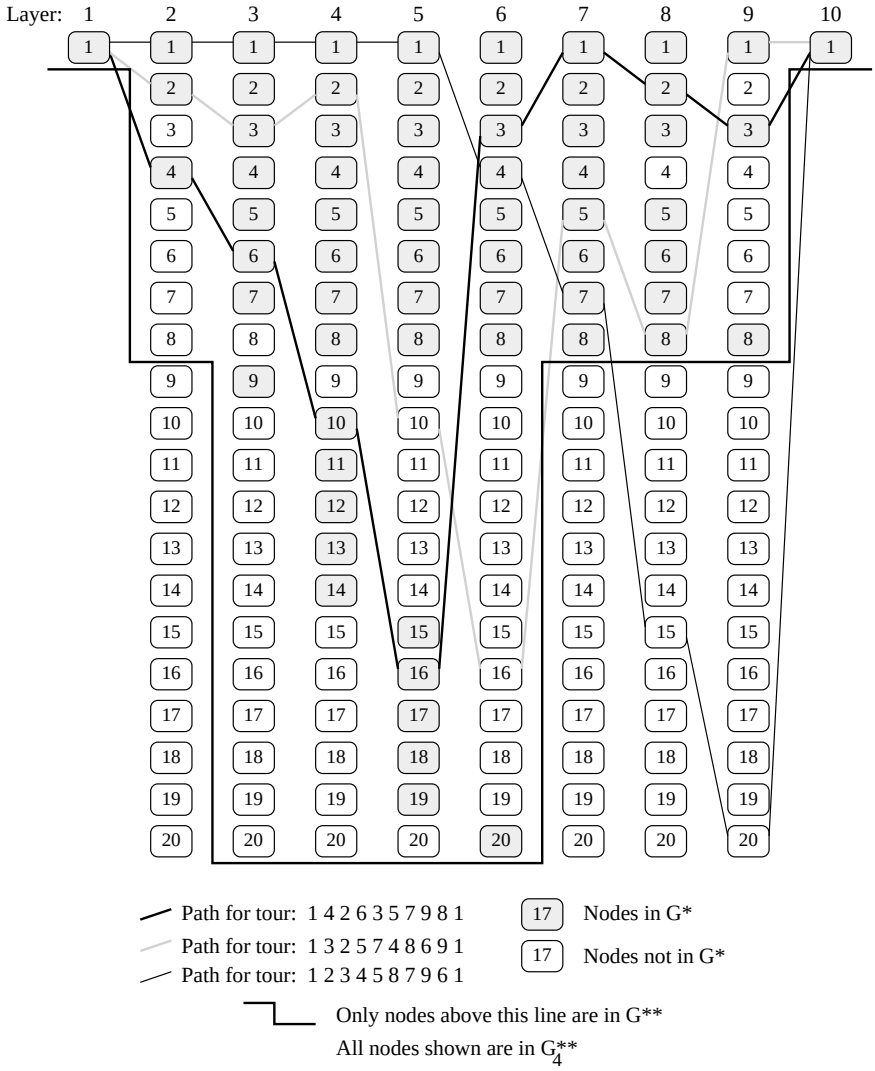

Fig. 2. G^*, G^{**}, and G_K^{**} for an example of problem 2

Proposition 3.1 *Given* $v := (j, S^-, S^+) \in W_K^*$,

$$L(v) = 1 + max\{|i - j|,\ max\{S^-\} - min\{S^+\},\ j - min\{S^+\}\}.$$

Proof: To show that $L(v) \geq 1 + j - min\{S^+\}$, consider the following: The state associated with v implies that city j is visited before any cities in S^+, and so $\pi(min\{S^+\}) > \pi(j)$; hence if $j \geq min\{S^+\} + L(v)$, then the precedence constraint would be violated; giving $L(v) > j - min\{S^+\}$. Since $L(v)$ is integer, this

is the same as $L(v) \geq 1 + j - min\{S^+\}$. The state associated with v also implies that all cities in S^- are visited before any cities in S^+, and so $\pi(min\{S^+\}) > \pi(max\{S^-\})$; hence if $max\{S^-\} \geq min\{S^+\} + L(v)$, then again the precedence constraint would be violated; giving $L(v) > max\{S^-\} - min\{S^+\}$. The state associated with v also indicates that city j is visited in the ith position. Since the precedence constraints imply that city j can only be visited in positions $j - L(v) + 1$ through $j + L(v) - 1$, we have $L(v) \geq 1 + |i - j|$, which now gives us

$$L(v) \geq 1 + max\{|i - j|,\ max\{S^-\} - min\{S^+\},\ j - min\{S^+\}\}.$$

To show equality, we simply build a tour whose associated path contains node v. This is done by first visiting the cities of $\{1, 2, \ldots, i-1\} \setminus S^+$ in increasing numeric order, then the cities of S^- in increasing order, then city j, then the cities of $S^+ \setminus \{j\}$ in increasing order, and finally the cities of $\{i, i+1, \ldots, n\} \setminus S^-$ in increasing order.□

The *depth* of layer i is the highest level of a node in W_K^* which appears in layer i of G^*. For the example in Figure 2, $D(1) = 1$, $D(2) = 3$, $D(3) = \ldots = D(6) = 4$, and so on.

The *reach* of a city $i \in N$ is the set of layers whose depth may be affected by $k(i)$. Designate this by $R(i)$. For the example in Figure 2, $R(3) = \{3, 4, 5, 6\}$. This is explained by the following proposition.

Proposition 3.2 *Given $i \in N$,*

$$R(i) = \{i, \ldots, i + k(i) - 1\}.$$

Proof: City i cannot be assigned position $i + k(i)$ or higher, otherwise some city $h \geq i + k(i)$ would be forced into one of the positions 1 through $i + k(i) - 1$. This would imply $\pi(i) > \pi(h)$, but since $h \geq i + k(i)$, this contradicts the precedence constraint. So the reach of i can go no further than position $i + k(i) - 1$. The reach does not fall below position i because the precedence constraints that allow city i to be visited in a position less than i involve only values of $k(j)$ for $j < i$.□

Proposition 3.3 *Given $i \in N$, the depth of layer i is no greater than*

$$D(i) := max_j\{k(j) : i \in R(j)\}.$$

Proof: The expression for $D(i)$ accounts for every city whose reach includes position i.□

We now construct the layered digraph G^{**} using $W^*_{D(i)}$ as the node set for layer i. G^{**} is an intermediate structure between G^* and G_K^{**}. Since the complexity of solving any particular problem instance is a linear function of the number of arcs in G^{**}, the following result is of particular interest:

Theorem 3.4 *The number of arcs of G^{**} is bounded by*

$$\sum_{i=2}^{n+1} D(i-1)(D(i)+1)2^{D(i)-2}.$$

Proof: The number of nodes in layer i of G^{**} is $(D(i)+1)2^{D(i)-2}$ (shown in [2]), so we need to show only that the maximum in-degree of a node in layer i of G^{**} is $D(i-1)$. In [2], it was also shown that for a given node $v := (j, S^-, S^+)$ in layer i of G^{**}, at most one node $u := (l, T^-, T^+)$ from layer $i-1$ may be a predecessor of v for a fixed value of l, so we need to show that there are at most $D(i-1)$ candidates for l. The depth of layer $i-1$ limits the candidates for l to the set $\{(i-1)-D(i-1)+1, \ldots, (i-1)+D(i-1)-1\}$. Also, by the compatibility of nodes u and v, the state associated with node v must restrict l to the set $(\{1, \ldots, i-1\} \cup S^-) \setminus S^+$. Intersecting these two sets, we conclude that l must be in the set $P := (\{i-D(i-1), \ldots, i-1\} \cup S^-) \setminus S^+$. Since $|S^+| = |S^-|$, $|P| = D(i-1)$ as long as $S^+ \subset \{i-D(i-1), \ldots, i-1\}$. If this were not the case, then $min\{S^+\} \leq (i-1)-D(i-1)$ since the elements in S^+ only come from the set of cities less than i, and so, for a node $u := (l, T^-, T^+)$ to be a potential predecessor of v from layer $i-1$, we must have $min\{T^+\} \leq min\{S^+\} \leq (i-1)-D(i-1)$, which would imply

$$L(u) \leq D(i-1) \leq (i-1) - min\{T^+\} \leq max\{T^-\} - min\{T^+\}$$

contrary to Proposition 3.1.□

This improves the bound given in [2],

$$\sum_{i=2}^{n} k^*(i-1)^2 k^*(i)^2 2^{k^*(i-1)+k^*(i)-2},$$

where $k^*(i) := max\{k(i), k(j_{(i)})\}$, and $j_{(i)} := min\{j : j + k(j) \geq i+1\}$, since the $D(i)$ are of the same order as the $k^*(i)$. Furthermore, the bound given in [2] requires an additional condition on the family $k(i)$, namely that $k(i)-k(i+1) \leq 1$ for each $i \in N$, a condition not needed for our result.

As illustrated by Figure 2, there may be paths in G^{**} which are not present in G^*. Thus we need a test for the nodes of G^{**} to prevent these paths from being considered.

The *k-threshold* for a node $v := (j, S^-, S^+) \in W_K^*$, denoted $kthresh(v)$, is the smallest value of $k(j)$ that permits the possibility of $v \in G^*$. Thus $k(j) < kthresh(v)$ implies $v \notin G^*$.

To calculate the k-threshold for a node $v := (j, S^-, S^+) \in W_K^*$, we notice that $v \in G^*$ implies that $k(j)$ is larger than the difference between j and all higher-numbered cities visited before j in the tour. The highest-numbered city visited before j is $max\{S^-\}$, unless S^- is empty, in which case no higher-numbered city can be visited before j. From this we have $k(j) > max\{0, max\{S^-\} - j\}$, so $kthresh(v) = 1 + max\{0, max\{S^-\} - j\}$.

Proposition 3.5 *Every path in G^{**} corresponding to an infeasible tour contains at least one node $v := (i, j, S^-, S^+)$, such that $kthresh(v) > k(j)$.*

Proof: Let π be the permutation for an infeasible tour T. Then there exist two cities, q and j, such that $\pi(j) > \pi(q)$ and $q \geq j + k(j)$. Let $i := \pi(j)$, (i.e. j is

in the ith position). Let $v := (j, S^-, S^+)$ be the node used in layer i of G^{**} for the path corresponding to T.

If $q \geq i$, then $q \in S^-$, and so:

$$\begin{aligned} kthresh(v) &= 1 + max\{0, max\{S^-\} - j\} \\ &\geq 1 + max\{S^-\} - j \geq 1 + q - j > q - j \geq k(j). \end{aligned}$$

If $q < i$, then $j < q < i$, and so $j \in S^+$. Since $|S^+| = |S^-|, S^-$ cannot be empty, and so $max\{S^-\} \geq i$ since the elements in S^- only come from the set of cities greater than or equal to i. This gives:

$$kthresh(v) = 1 + max\{0,\ max\{S^-\} - j\} \geq 1 + i - j > 1 + q - j > q - j \geq k(j). \square$$

In Figure 2, the k-threshold of node 16 in layer 6, which is 4, is higher than $k(j)$, which is 3. (j for node 16 in layer 6 is 4.)

Once W_K^* has been built for a certain K, *any problem instance with* $k(j) \leq K$ for all $j \in N$ can be solved to optimality (with a guarantee of optimality) by determining G^{**} and finding a shortest $s - t$ path in the subgraph G^* of G^{**}, where G^* is the auxiliary graph associated with the specific problem instance one wants to solve. Determinimg G^{**}, extracting the nodes and arcs of G^* from those of G^{**}, and putting the appropriate costs on the arcs does not increase the complexity of finding a shortest $s - t$ path in G^*, which remains linear in the number of arcs of G^{**}.

When applying this algorithm to time window problems, first an initial ordering based on sorting the cities by time window midpoint is constructed; then the implied precedence constraints are formed. While traversing the graph G^{**}, paths that correspond to infeasible solutions because of the time window restriction are weeded out by testing the cost at node $v := (j, S^-, S^+)$ (based on travel time) against the time window for city j. If we arrive at city j before the window opens, we pay the expense for waiting until the window opens. If we arrive at city j after the window closes, then any tour with a state associated with this node must be infeasible.

For the time window problems in [3], the objective was to minimize the total distance, not the total time, which makes a difference when tours must wait at a city for a time window to open. In this case, both the distance and the time must be kept at each node. However, this is not enough, since a tour that chooses to wait in one place to gain an advantage in distance may not be able to satisfy an important time window later.

Figure 3 illustrates this point. The route 1-2-3-4 has distance 6, but requires a time of 9 because there is a wait at city 2 This wait prevents the route from continuing to city 5 before its window closes. The route 1-3-2-4 has distance 8, but also has a total time of 8, so this route can continue through city 5. If the algorithm is not modified to keep at least the best *two* partial tours at each node in the auxiliary digraph, this solution will not be found.

In the case where the implied precedence constraints require a value of K too large to make the algorithm practical, the algorithm can still be run with a smaller value of K, but in this case, solutions found are not guaranteed to

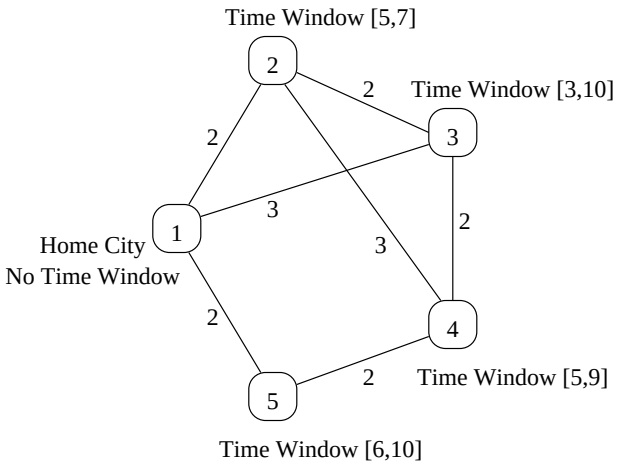

Fig. 3. Illustrating the difficulty of minimizing distance instead of time

be optimal. In such situations, using the initial sequence mentioned above, the algorithm does not perform well, but as the results show in Section 4, starting from the heuristic solution of another method, the algorithm can often improve the solution.

A variant of the time window problem, which we will call the *traveling salesman problem with time targets* (TSPTT), gives a target time for each city, rather than a window. The objective is to find a tour which minimizes the maximum deviation between the target time and actual service time over all cities. This algorithm can solve such problems by constructing windows of a fixed size d centered at each city's target time. A binary search is then used to find the smallest d, to a predetermined accuracy, that admits a feasible solution. Once the smallest such d is determined, the cheapest feasible tour is returned. Applications of this problem include delivery of perishable items (such as fresh fruit) to events without storage facilities, or routing of repair vehicles to customers with busy schedules who do not wish to wait for a potentially long period of time before being serviced.

4 Computational Experience

All of the results in this section were achieved on a Sun SparcStation 5.

Using a method similar to that given by Baker [1], Tama [10] created eight 20-city and five 26-city problems, representing one-machine scheduling problems with set-up times formulated as asymmetric TSP's with time windows, which he was unable to solve using cutting planes followed by a branch and bound algorithm, and which he kindly made available to us. We ran our algorithm on

Table 1.

	Problem Name	Required K	Solution Value (seconds) $K = 13$	$K = 17$
(20-city problems)	p192358	13	Infeasible* (1)	
	p192422	10	Infeasible* (1)	
	p192433	12	Infeasible* (1)	
	p192452	10	Infeasible* (1)	
	p192572	11	Infeasible* (1)	
	p192590	12	Infeasible* (1)	
	p193489	13	Infeasible* (1)	
	p194450	9	936* (1)	
(26-city problems)	p253574	16	Infeasible (1)	Infeasible* (7)
	p253883	18	1188 (2)	1188 (42)
	p254662	16	1140 (3)	1140* (23)
	p255522	13	1373* (2)	
	p256437	13	1295* (2)	

*indicates that a guarantee of optimality (or a guarantee of the infeasibility of the problem) was found.

these problems (using $K = 13$ and $K = 17$) with the following outcome: Of the eight 20-city problems, 7 were *proved* to be infeasible, and the remaining 1 was solved to optimality. Of the five 26-city problems, 1 was *proved* to be infeasible, 3 were solved to optimality, and the remaining 1 was solved without a guarantee of optimality because the required value of K was too high. (see Table 1)

As to symmetric TSP's with time windows, thirty problems based on the RC2 problems proposed by Solomon [9] that were first studied by Potvin et al. [6] and then Gendreau et al. [3] were run with our algorithm, using $K = 15$. The problem name listed in Table 2 refers to the problem code used by Solomon, followed by the route number. Neither Potvin nor Gendreau could guarantee the optimality of a solution even if their solution was optimal. The entry for "Required K" in Table 2 indicates what value of K would be needed for a guarantee of optimality.

Our algorithm solved 14 of these problems with a guarantee of optimality. Of these 14, Potvin had solved 2 to optimality and Gendreau had solved 9 to optimality. Of the remaining 16 problems, our algorithm fared better than Potvin 12 times, equalled Gendreau 2 times, and fared better than Gendreau 3 times. On the 8 problems where our algorithm had the most difficulty, we then used the solution generated by Gendreau as the initial tour, and we improved that solution in 5 of these 8 instances. Note that the solution we received from Gendreau was sometimes different than the solution portrayed in the results found in [3]. (see Table 2)

As mentioned above, the problems in Table 2 added an additional level of difficulty in that the objective was to minimize total distance rather than total time. For the reasons illustrated by Figure 3, we could not just keep the best partial solution at each node, but had to keep the best several partial solutions.

Table 2.

Problem	n	Potvin [6]	Gendreau [3]	Required K	q_0	q^*	Our Numbers (seconds)
rc201.1	20	465.53	444.54	6	1	5	444.54* (1) (1)
rc201.2	26	739.79	712.91	7	1	5	711.54* (1) (1)
rc201.3	32	839.76	795.44	7	2	7	790.61* (1) (1)
rc201.4	26	803.55	793.64	7	1	4	793.63* (1) (1)
rc202.1	33	844.97	772.18	20	3	>15	772.33 (88)
rc202.2	14	328.28	304.14	13	1	11	304.14* (1) (1)
rc202.3	29	878.65	839.58	15	2	11	837.72* (25) (60)
rc202.4	28	852.73	793.03	18	14	>15	804.57 (263)
rc203.1	19	481.13	453.48	13	2	21	453.48* (5) (14)
rc203.2	33	843.22	784.16	24	2	>15	832.14 (98)
with initial tour from [3]:			789.04	32	2	>15	784.16 (70)
rc203.3	37	911.18	842.25	27	2	>15	none found
with initial tour from [3]:			842.03	36	4	>15	834.90 (136)
rc203.4	15	330.33	314.29	12	1	11	314.29* (1) (3)
rc204.1	46	923.86	897.09	39	9	>15	none found
with initial tour from [3]:			878.76	45	1	>15	878.64 (92)
rc204.2	33	686.56	679.26	28	2	>15	683.13 (167)
with initial tour from [3]:			664.14	32	4	>15	664.14 (158)
rc204.3	24	455.03	460.24	22	4	>15	466.21 (183)
rc205.1	14	381.00	343.21	7	1	8	343.21* (1) (1)
rc205.2	27	796.57	755.93	11	1	14	755.93* (1) (2)
rc205.3	35	909.37	825.06	23	3	>15	825.06 (78)
rc205.4	28	797.20	762.41	12	6	11	760.47* (3) (5)
rc206.1	4	117.85	117.85	3	1	1	117.85* (1) (1)
rc206.2	37	850.47	842.17	16	3	>15	828.06 (66)
rc206.3	25	652.86	591.20	14	1	16	574.42* (7) (37)
rc206.4	38	893.34	845.04	16	2	>15	831.67 (54)
rc207.1	34	797.67	741.53	19	2	>15	743.28 (99)
rc207.2	31	721.39	718.09	21	5	>15	702.33 (239)
rc207.3	33	750.03	684.40	23	2	>15	719.66 (108)
with initial tour from [3]:			684.40	32	1	>15	684.40 (41)
rc207.4	6	119.64	119.64	5	1	5	119.64* (1) (1)
rc208.1	38	812.23	799.19	37	1	>15	834.64 (76)
with initial tour from [3]:			804.41	37	1	>15	795.58 (65)
rc208.2	29	584.14	543.41	28	3	>15	553.43 (118)
with initial tour from [3]:			543.41	28	2	>15	543.41 (94)
rc208.3	36	691.50	660.15	35	2	>15	676.25 (155)
with initial tour from [3]:			654.27	35	7	>15	649.11 (585)

Notes:
All problems run with $K = 15$.
*indicates a guarantee of optimality.
Times listed are for the best q. If a guarantee of optimality could be achieved, a second time listed is for the optimal q.

Table 3.

Problem Name	Open-Ended K&P	Our Results (seconds) $K = 10$	$K = 14$	$K = 17$
500.aa	2964	no improvement		
500.ba	7495	no improvement		
500.ca	21703	21690 (13)	21632 (290)	21618 (3641)
500.da	9950	9886 (14)	9857 (298)	9844 (3605)
500.ea	14724	no improvement		

Notes:
The Kanellakis & Papadimitriou heuristic was allowed to run for 1200 seconds.

This adds another dimension to the auxiliary graph, which we call the *thickness* of the graph, represented by the constant q, a bound on the number of partial solutions kept. Based on the problems in Table 2, the value of q required to find the best solution (q_0 in Table 2) was small, but the value of q required to guarantee a best solution for the given value of $K = 15$ (q^* in Table 2). was much larger

Five 500-city asymmetric TSP's were generated using the *genlarge* problem generator, which Repetto [8] kindly gave to us. Repetto solved these problems with an open-ended heuristic approach, where he would produce an initial tour by randomly choosing one of the nearest two neighbors, and then apply his implementation [7] of the Kanellakis & Papadimitriou heuristic [4], an adaptation to asymmetric TSP's of the Lin-Kernighan heuristic for the symmetric TSP [5]. This process would continue for 1200 seconds and the best solution found would be returned. The tours generated were used as initial tours for our algorithm, and our algorithm improved the solutions to 2 of these 5 problems. (see Table 3)

Table 4.

Number of Cities	Open-Ended K&P (seconds allowed)	$K = 8$ (seconds)	$K = 12$ (seconds)	$K = 15$ (seconds)	$K = 17$ (seconds)
200	233878 (300)	233878 (2)	233720 (19)	233720 (219)	233720 (1387)
300	272242 (600)	271597 (3)	271597 (29)	271597 (339)	271499 (2138)
400	295011 (1800)	294650 (5)	294504 (40)	294504 (449)	294443 (2928)
500	331354 (2400)	329727 (7)	329481 (52)	329371 (564)	329272 (3552)
600	351713 (3600)	351447 (9)	351318 (63)	351203 (692)	351126 (4293)
750	386920 (5500)	386457 (13)	385659 (84)	385659 (837)	385659 (5580)
1000	432558 (7500)	432329 (21)	431791 (166)	431736 (1131)	431094 (7364)

Notes:
The time shown for the Kanellakis & Papadimitriou heuristic is the running time allowed for the open-ended heuristic.

We built symmetric TSP's of various sizes using the t largest cities of the United States, for $t = 200, 300, 400, 500, 600, 750, 1000$. Population figures and coordinates were obtained from the United States Census Bureau [11]. Distances were calculated based on these coordinates, assuming a perfectly spherical earth with a radius of 6378.15 kilometers. All distances were rounded to the nearest tenth of a kilometer. Repetto then applied the above described open-ended heuristic [7] to these problems, and these tours were again used as initial tours for our algorithm. Our algorithm improved each of these solutions, usually with

Table 5.

		Required	Results for $K = 10$			Results for $K = 14$			Results for $K = 17$		
Problem	n	K	d	cost	sec.	d	cost	sec.	d	cost	sec.
rc201.1	20	5	81.92*	611.10	1						
rc201.2	26	6	99.56*	875.73	1						
rc201.3	32	5	93.53*	869.68	2						
rc201.4	26	7	114.50*	891.93	2						
rc202.1	33	≈19	277.38	875.97	7	254.08	864.32	139	254.08	864.32	1337
rc202.2	14	8	118.12*	552.18	2						
rc202.3	29	8	134.62*	889.27	4						
rc202.4	28	17	244.41	925.77	6	237.26	900.66	103	273.26*	900.66	871
rc203.1	19	14	220.23	610.81	3	220.23*	610.81	56			
rc203.2	33	≈25	387.57	974.42	8	387.57	974.42	174	383.87	972.58	1976
rc203.3	37	≈27	535.94	975.11	9	482.71	957.49	221	403.46	950.40	2452
rc203.4	15	11	211.98	598.25	2	211.98*	598.25	30			
rc204.1	46	≈43	770.78	1078.49	14	658.42	1022.31	335	623.43	966.05	4101
rc204.2	33	≈30	453.11	781.41	9	434.94	793.15	221	427.36	789.36	2332
rc204.3	24	≈23	321.52	659.22	7	321.52	659.22	136	321.52	659.21	1122
rc205.1	14	3	53.03*	473.12	1						
rc205.2	27	12	186.27	816.47	5	186.27*	816.47	72			
rc205.3	35	15	240.43	965.58	7	240.43	965.58	135	240.43*	965.58	1117
rc205.4	28	11	159.55	883.81	4	159.55*	883.81	64			
rc206.1	4	3	22.08*	227.04	1						
rc206.2	37	13	202.38	902.06	8	202.38*	902.06	136			
rc206.3	25	11	145.93	697.63	3	145.93*	697.63	51			
rc206.4	38	14	204.53	941.35	8	204.53*	941.35	156			
rc207.1	34	17	234.99	906.66	7	234.99	906.66	139	234.99*	906.66	1085
rc207.2	31	≈20	241.18	798.50	7	241.18	798.50	126	241.18	798.50	1348
rc207.3	33	≈18	275.88	896.67	8	259.51	880.36	163	259.51	880.36	1793
rc207.4	6	1	12.08*	309.85	1						
rc208.1	38	≈28	449.78	936.43	10	406.04	914.56	242	375.77	899.40	2703
rc208.2	29	≈21	301.41	779.39	6	267.46	762.42	113	267.46	762.42	1072
rc208.3	36	≈24	297.89	824.90	8	285.85	818.67	177	285.85	818.67	2082

Notes:
d is the value for the minimum window size returned by the algorithm.
*indicates a guarantee of optimality (within .01).

values of K as small as 8 (see Table 4). We suspect the high rate of success on these problems comes from the tendency for cities to cluster in metropolitan areas, which would tend to imply precedence constraints of the type (iii) outlined in problems 1 and 2.

To generate instances of the traveling salesman problem with time targets (TSPTT), we used the same data studied by Potvin [6] and Gendreau [3], and used the time window midpoints for the time targets. The results are shown in Table 5. In many cases, the optimal solution was found with a value of K much smaller than that needed to guarantee optimality, which may indicate that some solutions given to problems without a guarantee of optimality are optimal. In cases where no guarantee was achieved, the exact value of K needed for a guarantee is not known. Arc costs for the problems in Table 5 generally ranged from 10 to 100 units, and 1 time unit is required to travel one distance unit.

References

1. E. Baker, "An Exact Algorithm for the Time-Constrained Traveling Salesman Problem." *Operations Research, 31,* (1983) 938-945.
2. E. Balas, "New Classes of Efficiently Solvable Generalized Traveling Salesman Problems," Management Science Research Report #MSRR-611, Graduate School of Industrial Administration, Carnegie Mellon University, March 1995.
3. M. Gendreau, A. Hertz, G. Laporte, M. Stan, "A Generalized Insertion Heuristics for the Traveling Salesman Problem with Time Windows." Publication CRT-95-07, Centre de recherche sur les transports, Montréal, January 1995.
4. P. Kanellakis, C. Papadimiriou, "Local Search for the Traveling Salesman Problem." *Operations Research, 28,* (1980) 1086-1099.
5. S. Lin, B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem." *Operations Research, 21,* (1973) 495-516.
6. J.-Y. Potvin, S. Bengio, "A Genetic Approach to the Vehicle Routing Problem with Time Windows." Publication CRT-953, Centre de recherche sur les transports, Montréal, 1993.
7. B. Repetto *Upper and Lower Bounding Procedures for the Asymmetric Traveling Salesman Problem.* Ph.D. Thesis, GSIA, Carnegie Mellon University, April 1994.
8. B. Repetto, personal communication.
9. M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling with Time Windows Constraints." *Operations Research, 35,* (1987) 254-265.
10. J. Tama, personal communication.
11. United States Census Bureau, http://www.census.gov/cgi-bin/gazetteer

This article was processed using the LaTeX macro package with LLNCS style