Submitted manuscript

A Bi-directional Resource-bounded Dynamic Programming Approach for the Traveling Salesman Problem with Time Windows

Jing-Quan Li
California PATH, University of California, Berkeley, Richmond, CA 94804, jingquan@path.berkeley.edu

This paper presents a bi-directional resource-bounded label correcting algorithm for the traveling salesman problem with time windows, in which the objective is to minimize travel times. Label extensions and dominance start simultaneously in both forward and backward directions: the forward direction from the starting depot and the backward direction from the terminating depot. The resultant label extension process scans much smaller the space than in single directional dynamic programming, substantially reducing the number of non-dominated labels. The labels for both the forward direction and backward direction are ultimately joined to form a complete route if all relevant feasibility conditions are satisfied. Our algorithm generates optimal solutions for a number of the instances with wide time windows for which no optimal solutions have been previously reported.

Key words: Dynamic Programming, Time Windows, Bi-directional Search

1. Introduction

As one of the most well-studied combinatorial optimization problems, the traveling salesman problem (TSP) can simply be stated as follows: if a traveling salesman starts from the depot, visits n customers exactly once each, and returns to the depot, what is the least costly route? The traveling salesman problem has many variations. Among them, the traveling salesman problem with time windows (TSPTW) requires that customers be served within a given set of time windows. If the salesman arrives before the time window, the salesman has to wait before serving the customer. Service times may be also required for each customer visit.

Applications of the TSPTW include machine scheduling and automatic guide vehicle routing in flexible manufacturing systems (Gendreau et al. 1998). The TSPTW also arises as a sub-problem in the vehicle routing context if a cluster-first route-second approach is employed (Gendreau et al. 1998). In addition, the TSPTW can be transformed into a separation subproblem when a branch-and-cut approach is used to solve vehicle routing with time windows (Kohl et al. 1999, Desaulniers et al. 2008).

The TSPTW can be formally defined as follows. Let $N = \{1, 2, \ldots, n\}$ be the set of customers that the traveling salesman will visit. Let 0 refer to the depot as the starting point and $n+1$ be the depot as the terminating point. Let c_{ij} be the cost of traveling from customer i to customer j. The time window of customer i is $[a_i, b_i]$. The service time of customer i is W_i. Define a binary variable x_{ij} to equal 1 if the salesman visits customer j directly after visiting customer i, and 0 otherwise. Define s_i as the time when the service starts at customer i. The TSPTW is then formulated as

$$
\begin{array}{lll}
\min \sum_{i=0}^{n} \sum_{j=1}^{n+1} c_{ij} x_{ij} & & (a) \\
\text{st} & & \\
\sum_{j=1}^{n+1} x_{ij} = 1, & i = 0, 1, \ldots, n & (b) \\
\sum_{i=0}^{n} x_{ij} = 1, & j = 1, 2, \ldots, n+1 & (c) \\
x_{ij}(s_i + W_i + t_{ij} - s_j) \leq 0, & i = 0, 1, \ldots, n, j = 1, 2, \ldots, n+1 & (d) \\
a_i \leq s_i \leq b_i, & \forall i = 1, 2, \ldots, n+1 & (e) \\
s_0 = 0 & & (f) \\
x_{ij} \in \{0, 1\} & i = 0, 1, \ldots, n, j = 1, 2, \ldots, n+1. &
\end{array}
$$

Objective (a) is to minimize the total travel cost. Constraints (b) and (c) ensure that each customer is served once and only once. Constraint (d) guarantees that if the salesman serves customer j immediately after visiting customer i, the starting time at j will be no earlier than the time of arrival from i. Constraint (e) ensures that the service starting time is within the given time window. Finally, constraint (f) requires that the salesman starts from the depot at time 0. Note that the TSPTW objective in this paper is to minimize the total arc traversal cost instead of minimizing the time to return to the depot. Savelsbergh (1985) proves that even finding a feasible solution for the TSPTW is NP-hard.

Exact solution approaches to the TSPTW include the branch-and-bound algorithm (Baker 1983), the two-commodity flow formulation with the branch-and-bound scheme (Langevin et al. 1993), the dynamic programming with label reduction techniques (Dumas et al. 1995), and the constraint programming (Pesant et al. 1998, Focacci et al. 2002). These exact approaches generally provide TSPTW solutions with relatively narrow time windows. Heuristics methods include the generalized insertion heuristic with local re-optimization (Gendreau et al. 1998), the constraint programming (Pesant et al. 1999), the construction heuristic with local search improvements (Calvo 2000) and the compressed-annealing heuristic (Ohlmann and Thomas 2007).

As shown in Dumas et al. (1995), dynamic programming (DP) with label reduction techniques is an effective approach to solve the TSPTW optimally if the time window is relatively narrow. Dumas et al. (1995) solved various TSPTW problems, including instances with 200 customers and time windows of 40 units and instances with 60 customers and time windows of 100 units. However, to our knowledge, no optimal solutions have been found for TSPTW instances with large time windows, such as those proposed by Gendreau et al. (1998).

Dynamic programming has also been widely used to solve the resource-constrained shortest path problem (for an example, see Desrochers and Soumis (1988)). Recently, bi-directional resource-bounded dynamic programming approaches have been proposed by Righini and Salani (2006, 2008) and Boland et al. (2006) to solve the elementary shortest path problem with resource constraints, a NP-hard problem in the strong sense. Righini and Salani (2006, 2008) and Boland et al. (2006) report successful solutions to certain elementary shortest path problems using this strategy.

Motivated by the effectiveness of bi-directional dynamic programming in solving the resource-constrained shortest path problem, we designed a bi-directional resource-bounded dynamic programming algorithm to solve the TSPTW with relatively wide time windows. Algorithm design is described in section 2. Computational experiments are presented in section 3. Section 4 concludes the paper.

2. Bi-directional Resource-bounded Label Correcting Algorithm

The key ideas behind bi-directional resource-bounded dynamic programming are as follows: (i) label extension is conducted from both the starting and terminating depot at the same time: one operation begins at the starting depot and moves toward the terminating depot, while the other operation begins at the terminating depot and moves toward the starting depot; (ii) for label extension in either direction, resource consumption cannot exceed half of the total available

resource; and (iii) labels generated in each of the two directions will be connected only if all relevant feasibility conditions are satisfied. The resource may include the vehicle capacity, total traversal time, etc. The bi-directional resource-bounded scheme significantly reduces the number of non-dominated labels since a label is not permitted to extend if its resource consumption exceeds half of the total resource availability.

2.1. Label Extension

First, we will discuss forward extension, where the path starts at node 0, namely the starting depot. Each state in the forward extension context is represented by a label, (V, s, i), where i is the last reached node and s represents the *earliest* time when service can commence at i. As suggested by Beasley and Christofides (1989), an additional resource, V, can be included to indicate if every node $i \in N \cup \{0, n+1\}$ is visited by this label. V can be implemented as a binary vector: initially, all the positions in V are set to 0; and if a node is visited, the corresponding position in V is set to 1. Alternatively, V can also be implemented to contain information regarding visit sequences. Let V_i denote the visit sequence of node i. Initially, $V_i = 0, i \in N \cup \{0, n+1\}$. If node i is the k^{th} node to get visited in the label, then $V_i = k$. The second approach is used in our study. The cost of label (V, s, i) is $c(V, s, i)$, representing the accumulative travel cost from the starting depot. The first label in the forward extension is $(V, 0, 0)$, which starts at time 0 with cost 0. $V_0 = 1$ and $V_i = 0, i \in N \cup \{n+1\}$.

Now consider that label (V, s, i) is forward extended to label $(V^{'}, s^{'}, j)$. Label $(V^{'}, s^{'}, j)$ is determined as follows. $s^{'} = \max\{s + W_i + t_{ij}, a_j\}$ since the salesman has to wait if the arrival time to node j is before the time window. $V_k^{'} = V_k$ if $k \neq j$, and $V_j^{'} = V_i + 1$. The extension is feasible if (i) $s^{'}$ is within the time window$(a_j \leq s^{'} \leq b_j)$ and (ii) j has not previously been visited $(V_j = 0)$. The extension can also be strengthened by examining if the new label, $(V^{'}, s^{'}, j)$, can be further extended to visit all the non-visited nodes while still satisfying the relevant time window constraints. If not, the extension from (V, s, i) to $(V^{'}, s^{'}, j)$ is not feasible. Dumas et al. (1995) report that this test (referred to as Test 2 in their paper) is very effective in reducing the number of labels.

As suggested in Righini and Salani (2006), we can determine the maximum feasible arrival time to the terminating depot (node $n+1$), say $T = \max_{i \in N}\{b_i + W_i + t_{i,n+1}\}$. In the backward extension context, we start from node $n+1$ and we extend backward to other nodes. The *latest* time when the service can start at a node is determined such that the final arrival time to the terminating depot is not later than T. The label in the backward extension is represented by (V, s, i), where V and i have same definitions as for the forward extension. s is the *latest* time when the service can start at node i. The first label of the backward extension is $(V, T, n+1)$, where $s = T$ and $i = n+1$. $V_{n+1} = 1$ and $V_i = 0, i \in N \cup \{0\}$.

Consider that label (V, s, i) is backward extended to $(V^{'}, s^{'}, j)$. If $s^{'}$, the latest time when service can start at node j, is later than the time window $(s^{'} > b_j)$, any starting time within the window is feasible. Hence, we set $s^{'} = \min\{s - W_j - t_{ji}, b_j\}$ since service cannot be started after b_j. $V_k^{'} = V_k$ if $k \neq j$, and $V_j^{'} = V_i + 1$. The backward extension operation is feasible if (i) $s^{'}$ is no later than the time window $(s^{'} \geq a_j)$, (ii) j has not previously been visited $(V_j = 0)$, and (iii) label $(V^{'}, s^{'}, j)$ can be further extended to visit all the non-visited nodes in $(V^{'}, s^{'}, j)$ while still satisfying the time window constraints.

Note that in the context of backward extensions, Righini and Salani (2006) use τ^{bw} as the minimum time that must be consumed since the departure from node j such that the final arrival time to the terminating depot is never later than T. In our study, we use the latest time when the service can start at node j, which is in fact complementary to τ^{bw}. We believe that the latest time when the service can start is easier to understand and corresponds to the earliest time when the service can start in the traditional forward extension.

2.2. Label Dominance

As pointed out by numerous researchers, reducing the number of labels is crucial to design an efficient dynamic programming algorithm. Label reductions can be achieved by limiting the extension as well as by removing dominated labels.

Assume in the case of forward extensions that both label (V, s, i) and label $(\bar{V}, \bar{s}, i)$ have node i as their last reached node. Define set $U = \{i|V_i \neq 0, i \in N \cup \{0, n+1\}\}$ and set $\bar{U} = \{i|\bar{V}_i \neq 0, i \in N \cup \{0, n+1\}\}$ as the nodes that have been visited by labels (V, s, i) and $(\bar{V}, \bar{s}, i)$ respectively. Label (V, s, i) dominates label $(\bar{V}, \bar{s}, i)$ if (i) $s \leq \bar{s}$, (ii) $c(V, s, i) \leq c(\bar{V}, \bar{s}, i)$, and (iii) $U = \bar{U}$. However, if the triangular inequality ($c_{ij} + c_{jk} > c_{ik}$ for customers i, j, and k, where c_{ij} represents the cost traveling from i to j) holds in the underlying network, the resulting dominance can be strengthened as follows: label (V, s, i) dominates label$(\bar{V}, \bar{s}, i)$ if (i) $s \leq \bar{s}$, (ii) $c(V, s, i) \leq c(\bar{V}, \bar{s}, i)$, and (iii) $\bar{U} \subseteq U$. The notion of the triangular inequality has also been used to strengthen dominance rules in the resource-constrained shortest path problem (Feillet et al. 2004).

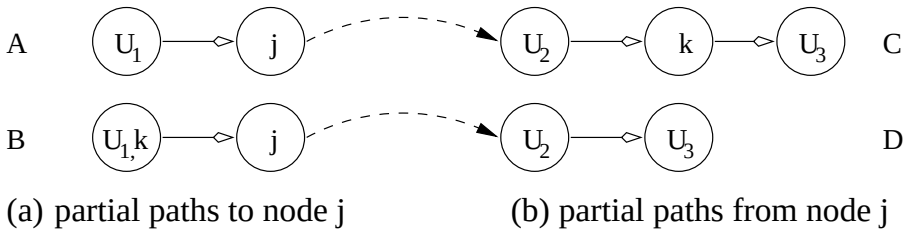

Figure 1 **An Example of the Triangle Inequality**

Figure 1 gives an example of the triangular inequality, where partial path A can be visited by label $(\bar{V}, \bar{s}, i)$ with $\bar{U} = \{U_1, j\}$, and partial path B can be visited by (V, s, i) with $U = \{U_1, k, j\}$. Note that $\bar{U} \subseteq U$. In addition, assume $s \leq \bar{s}$ and $c(V, s, i) \leq c(\bar{V}, \bar{s}, i)$. First, partial path A can be concatenated with partial path C, while partial path B cannot be joined with partial path C since node k has already been visited in partial path B. However, after node k is removed from partial path C, the cost of resultant partial path D is guaranteed to be smaller than the cost of partial path C if the triangular inequality holds. Hence, together with $s \leq \bar{s}$ and $c(V, s, i) \leq c(\bar{V}, \bar{s}, i)$, it is clear that (V, s, i) dominates $(\bar{V}, \bar{s}, i)$.

However, if the triangular inequality does not hold, the cost of partial path D is not guaranteed to be smaller than the cost of partial path C. $\bar{U} = U$ has to hold instead of $\bar{U} \subseteq U$. The computational time can be substantially reduced consistent with the triangular inequality, a fact that will be seen in our computational results.

In the case of backward extensions, dominance rules (i) and (iii) remain the same as those in the forward extension. However, s and $\bar{s}$ are opposite. Label (V, s, i) dominates label $(\bar{V}, \bar{s}, i)$ if (i) $s \geq \bar{s}$, (ii) $c(V, s, i) \leq c(\bar{V}, \bar{s}, i)$, and (iii) $U = \bar{U}$. $U = \bar{U}$ can be replaced by $\bar{U} \subseteq U$ if the triangular inequality holds.

2.3. Resource Bounding and Label Joining

Label reductions can be strengthened significantly by resource bounding in the context of the bi-directional dynamic programming framework. Righini and Salani (2006) propose the following approach: in the forward extension from (V, s, i) to $(\bar{V}, \bar{s}, i)$, the extension is permitted only if $\bar{s} \leq T/2$; and in the backward extension from (V, s, i) to $(\bar{V}, \bar{s}, i)$, the extension is allowed only if $\bar{s} \geq T/2$ (more precisely, the minimum time which must be consumed since the departure from a node needs to smaller or equal to $T/2$). Intuitively speaking, each directional extension only reaches approximately half of the route. The whole route can be obtained by joining partial routes from the forward and backward extensions.

The approach proposed by Righini and Salani (2006) attempts to join a forward label of node i with a backward label of node j if there is a connection from node i to node j. In our TSPTW study, we attempt to join a forward label and a backward label for the same node since it is not necessary to scan the network. However, joining a forward and a backward label from the same node may result in a potential problem, as can be seen in Figure 2. In the case of the forward extension, the label with node i as the last reached node cannot be extended to node j, since the starting time of the new label will be larger than $T/2$. Similarly, in the case of the backward extension, the label with node j as the last reached node cannot be extended to node i, since the starting time of the new label will be smaller than $T/2$. Therefore, the optimal solution can never be reached. This potential problem is not present in the approach by Righini and Salani (2006) since under that framework a forward label of node i is joined to a backward label of a neighboring node of node i instead of node i itself.

Observe that the forward label at node i can be further extended to reach node j or otherwise that the backward label at node j can be extended to reach node i. In such cases we can guarantee that the optimal solution will not be missed. Let G be the maximum arc traversal time in the underlying network, in which case we can say that the optimal solution will not be missed if (i) in the forward extension to $(\bar{V}, \bar{s}, i)$, the extension is allowed only if $\bar{s} \leq T/2$; and in the backward extension to $(\bar{V}, \bar{s}, i)$, the extension is allowed only if $\bar{s} \geq T/2 - G$; or (ii) in the forward extension to $(\bar{V}, \bar{s}, i)$, the extension is allowed only if $\bar{s} \leq T/2 + G$; and in the backward extension to $(\bar{V}, \bar{s}, i)$, the extension is allowed only if $\bar{s} \geq T/2$. Approach (i) is used in our implementation.

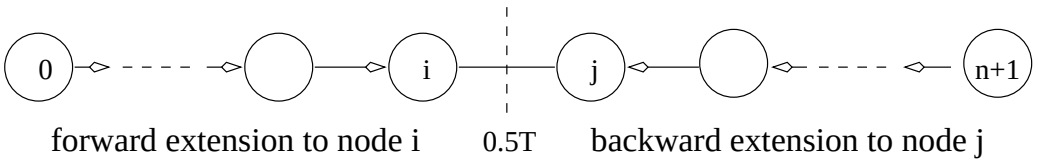

Figure 2 **A Potential Problem of Label Joining**

When a forward label and a backward label are joined, we need to investigate if all the nodes are covered by the combined label and if the starting time for the forward extension is less than or equal to the starting time associated with the backward extension. If either condition is not satisfied, the joining is infeasible.

An example is used to illustrate the bi-directional resource-bounded dynamic programming for the TSPTW problem. Figure 3 presents an underlying network for a TSPTW problem that features 5 customers. Nodes 0 and 6 reference the starting and terminating depot respectively. Service time at each node is 0. The travel time, travel cost and time window are evident in the figure.

The maximal arrival time to the terminating depot, T, is 10. The maximum arc traversal time is 2. Hence, in the case of forward extensions, the label starting time needs to be less than or equal to $10/2$ (i.e., 5); in the case of backward extensions, the label starting time must be greater than or equal to $10/2 - 2$ (i.e., 3). In Table 1, we present an overview of the bi-directional resource-bounded dynamic programming process. After the forward and backward extensions are complete, Table 2 describes the process that is used to join the labels.

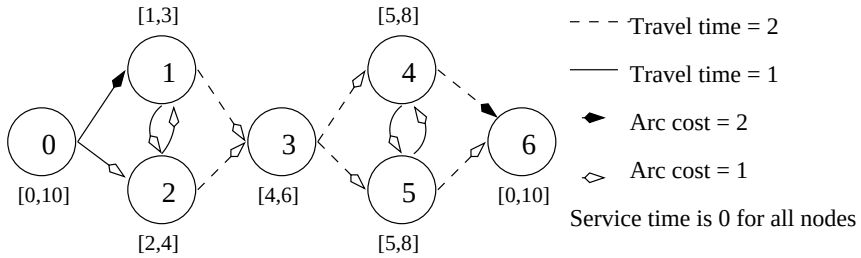

Figure 3 An example of bi-directional resource-bounded DP

Table 1 Labels in the Dynamic Programming Example

Forward Extension (Starting Time ≤ 5)					Backward Extension (Starting Time ≥ 3)				
Visit Sequence	Cost	Earliest Starting Time	Name	Previous Label	Visit Sequence	Cost	Latest Starting Time	Name	Previous Label
0	0	0	A		6	0	10	Z	
0-1	2	1	B	A	6-4	2	8	Y	Z
0-2	1	2	C	A	6-5	1	8	X	Z
0-1-2	3	2	D	B	6-4-5	3	7	W	Y*
0-2-1	2	3	E	C	6-5-4	2	7	V	X
0-1-2-3	4	4	F	D	6-5-4-3	3	5	U	V
0-2-1-3	3	5	G	E	6-5-4-3-2	4	3	T	U
					6-5-4-3-1	4	3	S	U

*: Label Y is dominated by label V and is dropped.

Table 2 Label Joining in the Dynamic Programming Example

Forward Labels		Backward Labels		After Joining				
Visit Sequence	Name	Visiting Sequence	Name	Final Route	Every node covered only once?	Earliest Starting v.s. Latest Starting	Total Cost	Note
0-1-2-3	F	6-5-4-3	U	0-1-2-3-4-5-6	Yes	$4 \leq 5$	$4+3=7$	
0-2-1-3	G	6-5-4-3	U	0-2-1-3-4-5-6	Yes	$5 \leq 5$	$3+3=6$	Optimal
0-1-2	D	6-5-4-3-2	T	0-1-2-3-4-5-6	Yes	$2 \leq 3$	$3+4=7$	
0-2-1	E	6-5-4-3-1	S	0-2-1-3-4-5-6	Yes	$3 \leq 3$	$2+4=6$	Optimal

The overall algorithm is outlined as follows.

Bi-directional Resource-bounded Label Correcting Algorithm for the TSPTW

Step 1: Place the first label for the forward extension and the first label for the backward extension into the active label pool.

Step 2: For all the labels in the active label pool:

Step 2.1: Remove the current label from the active label pool.

Step 2.2: For all the out-going arcs from the node associated with the current label:

Step 2.2.1: Determine if a new label can be extended from the current label given the relevant time windows, reachability of non-visited nodes and the relevant starting time thresholds.

Step 2.2.2: If the new label is generated and not dominated, insert it into the active label pool. Assess whether the new label can dominate other existing labels of the corresponding node. If so, delete these dominated labels.

Step 3: For each node, examine if a forward label can be joined to a backward label to form a complete label such that all the nodes are visited and all time windows are satisfied. If a feasible joining is found, store the combined route.

Step 4: The feasible route from Step 3 with the minimum cost is outputted as the optimal route.

2.4. Implementation Issues

Label pool management is very important since the most major operations generally involve label extensions and dominance. When the program starts, we provide a large memory allocation to hold the free label pool. If a new label is generated, the free label pool returns an available label.

Assume that label A is generated and is extended to label B. In subsequent operations, label A is dominated and dropped. However, label B may be still non-dominated. In the joining phase, we may process label B and may need to trace its predecessor node, i.e., the node with which label A is associated. However, label A has already been dropped and is now inaccessible. Therefore, a potential problem arises. Two approaches can be used to handle this problem. The first approach is to store dominated labels in a trash pool and not to delete information regarding the dominated labels, thereby making dominated labels accessible on demand. The second approach is to directly use the visit sequence information stored in V of label B. We use the latter approach in our implementation.

When a forward label and a backward label are joined together, we need to investigate if (i) the starting time at the forward label is less than or equal to the starting time of the backward label and (ii) if all the nodes are visited once and only once in the joined label. Checking starting times can be easy and fast. However, the node covering checking may be time consuming if it is not implemented efficiently. Our approach is to first obtain the numbers of nodes that have been visited in the forward label and backward label, say F and B. If $F+B$ is not equal to $n+3$ (n customers, one starting depot, one terminating depot and one common node between the two labels), the joined label is infeasible and we do not need to continue to check if all the nodes are covered once and only once in the joined label. For a forward label with F visited nodes, we do not need to scan the entire set of backward labels to search for labels with $B = n+3-F$. Instead, for each node, we can use the quicksort algorithm to sort the forward and backward labels according to the number of visited nodes. Then, for a forward label that exhibits F visited nodes, we can use the binary search to determine the potential search range in the backward label given $B = n+3-F$. The exact covering is only examined for this potential range, a fact that can reduce the computation time considerably.

3. Computational Experiments

This section presents our computational experiments for assessing the performance of the bi-directional resource-bounded label correcting algorithm.

3.1. Benchmark Problems

Four benchmark data sets are widely used in the literature.

Data set 1: problems proposed by Langevin et al. (1993)
Each problem with the same number of customers and time window widths has 10 instances. A travel-time matrix with fractional values is available. However, the triangular inequality may not always hold.

Data set 2: problems proposed by Solomon (1987)
These problems are obtained by Potvin and Bengio (1996) as individual routes from Solomon's RC2 instances (Solomon 1987). The instances feature x-y coordinates and the travel time is calculated as the fractional Euclidean distance using x-y coordinates. Since the travel time is fractional, the triangular inequality always holds.

Data sets 3 and 4: problems proposed by Dumas et al. (1995) and Gendreau et al. (1998)
Each problem with the same number of customers and time window widths has 5 instances. No travel-time matrix is directly available. Travel times have to be calculated from x-y coordinates by using the Euclidean distance model. In addition, integral travel times are used in most studies by truncating and adjusting the fractional values (Dumas et al. 1995, Gendreau et al.

1998, Calvo 2000). However, after a fractional distance has been truncated, the triangular inequality may no longer hold.

Certain studies attempt to maintain the validity of the triangular inequality by adjusting some travel time entries. However, the means to do this is unclear. Ohlmann and Thomas (2007) then propose a procedure that involves (i) calculating the Euclidean distance between node i to node j; (ii) adding the service time at node i into the travel time from i to j; (iii) truncating the travel time by using the floor operator; and (iv) adjusting the travel time for maintaining the validity of the triangular property. Their method is copied as follows (please see Thomas (2008) for further detail).

```
for (u32 i = 0 ; i <= numCustomers - 1; i++)
    for (u32 j = 0 ; j <= numCustomers - 1 ; j++)
        for (u32 k = 0 ; k <= numCustomers - 1 ; k++)
            if (travelTimes[i][j] > (travelTimes[i][k] + travelTimes[k][j]))
                travelTimes[i][j] = travelTimes[i][k] + travelTimes[k][j];
```

We note that using different orders of i, j and k in travelTimes may lead to different travelTimes entries. As mentioned by Thomas (2008), the triangular inequality is not guaranteed to be maintained by this procedure, an outcome that is confirmed in our experiments. The triangular property can be achieved by repeatedly calling the above procedure until no further violation is identified.

We generate two types of travel-time matrix for data sets 3 and 4. The first data type is determined by the procedure listed in Thomas (2008) and presented above, under which the triangular inequality may not always hold. We note that this way to generate the travel-time may be slightly different from those used in Dumas et al. (1995), Gendreau et al. (1998) and Calvo (2000). The corresponding travel-time matrix may be slightly different. The second type ensures the triangular inequality holds true by repeatedly applying the procedure recommended by Thomas (2008).

Ohlmann and Thomas (2007) also proposed some additional instances with large time windows. The entire data set is available on the website of Dr. Thomas (Thomas 2008). Most instances by Dumas et al. (1995) and Langevin et al. (1993) have relatively narrow time windows and have already been solved to optimality. Instances proposed by Gendreau et al. (1998) and Solomon (1987) exhibit wide time windows; to the best of our knowledge, no optimal solutions have been reported for these instances.

For the instances where the triangular inequality holds, the strengthened dominance rules are applied. Otherwise, the regular dominance rules are used. Since our goal in this paper is to find optimal solutions, we do not focus on comparing our solutions with those generated by heuristic algorithms.

The DP algorithm was implemented in C++ on a Lenovo Thinkstation with 4 Intel processors at 2GHZ, 4GB of RAM and a Linux operating system. Some instances suggested by Gendreau et al. (1998) were also solved using the mixed integer program (MIP) solver in CPLEX 11 on Sun Workstations, each of which with 2 UltraSPARCIII processors at 750MHz, 4GB of RAM and the Solaris 9 operating system. We note that all the algorithms are sequential and do not take advantage of the availability of multiple processors.

3.2. Computational Results

We use the following notation: n (number of customers), w (width of time windows), Inst (instance number), Opt (optimal objective), L (total number non-dominated labels), and CPU (CPU seconds by default).

Table 3 lists computational results of certain instances by Gendreau et al. (1998). The table shows the optimal objective, the number of non-dominated labels and the CPU time in seconds for each individual instance together with average values for instances with the same number of customers

Table 3 Optimal Solutions for Certain Instances by Gendreau et al. (1998)

n	w	Inst	With Triangle			Without Triangle			n	w	Inst	With Triangle			Without Triangle		
			Opt	$\|L\|$	CPU	Opt	$\|L\|$	CPU				Opt	$\|L\|$	CPU	Opt	$\|L\|$	CPU
20	120	1	267	2644	0.19	267	7059	0.17	20	140	1	176	5629	0.42	176	12920	0.46
		2	218	5910	0.43	218	12491	0.8			2	272	1903	0.05	272	2871	0.04
		3	303	1075	0.05	303	2968	0.06			3	236	2997	0.11	236	5665	0.13
		4	300	1118	0.03	300	1608	0.03			4	255	5583	0.35	255	10763	0.3
		5	240	3985	0.29	240	9996	0.51			5	225	2312	0.13	225	6396	0.19
		Avg	265.6	2946	0.20	265.6	6824	0.31			Avg	232.8	3685	0.21	232.8	7723	0.22
20	160	1	241	9041	1.68	241	24170	1.59	20	180	1	253	6722	0.45	253	11907	0.36
		2	201	3048	0.14	201	7268	0.17			2	265	4477	0.34	265	11030	0.54
		3	201	2607	0.15	201	6142	0.13			3	271	7545	0.5	271	12193	0.41
		4	203	9583	3.56	203	45920	4.46			4	201	16290	3.26	201	40346	3.23
		5	245	5525	0.32	245	10265	0.39			5	193	46879	29.9	193	143397	41.71
		Avg	218.2	5961	1.17	218.2	18753	1.35			Avg	236.6	16383	6.89	236.6	43775	9.25
20	200	1	233	29544	12.71	233	80635	14.5	40	120	1	434	16130	12.88	434	74689	36.94
		2	203	94645	68.53	203	209090	102.6			2	444	54838	54.78	444	183273	64.55
		3	249	10808	4.66	249	48490	5.09			3	357	37744	39.97	357	168974	48.75
		4	293	19804	3.5	293	39122	3.23			4	303	26706	32.98	303	157549	98.75
		5	227	66447	30.41	227	135490	39.25			5	350	9891	2.57	350	33835	2.85
		Avg	241.0	44250	23.96	241.0	102565	32.93			Avg	377.6	29062	28.64	377.6	123664	50.37
40	140	1	328	60599	46.85	328	217416	83.83	40	160	1	348	104615	117.30	348	359288	196.12
		2	383	92149	94.6	383	318629	199.46			2	337	213833	1331.49	337	879802	3727.44
		3	398	18693	5.68	398	59133	5.75			3	346	85982	474.49	346	616596	1456.60
		4	342	142454	249.42	342	459614	452.46			4						
		5	371	224708	1347.49	371	941226	3642.28			5	315	332075	9369.08	315	1883813	21653.18
		Avg	364.4	107721	348.81	364.4	399204	876.76			Avg						
60	120	1	384	504044	8431.44	384	2761089	27559.32	60	140	1						
		2	426	126114	419.32	427	603885	1278.86			2	462	251631	4497.57	462	2153415	15547.68
		3	407	461281	20411.30	407	3512238	72153.44			3						
		4	490	310446	13702.24	490	2525574	34152.45			4						
		5	547	185096	1455.13	547	1182566	5030.21			5	460	267884	5148.52	460	1916846	18289.83
		Avg	450.8	317396	8883.89	451.0	2117070	28034.86			Avg						

and time window widths. We solve Gendreau's instances involving 20 customers for all the time windows and with 40 customers with time windows of 120, 140, and 160 units. We also solve certain instances involving 60 customers with time windows of 120 and 140 units. Computational time for situations with the triangular inequality is significantly smaller than for situations without the triangular inequality (see columns 6, 9, 15 and 18 for details) since the stronger dominance rule can be applied if the triangular inequality holds.

In order to verify our algorithm, we also use the MIP solver in CPLEX 11 to solve instances involving 20 customers with variable time windows and 40 customers with time window fixed to length 120 units. We consider only those instances without the triangular inequality. Our optimal solutions match those of CPLEX very well (see columns 4, 6, 11, 13, 17 and 19 in Table 4). Although CPLEX ran on a relatively slow machine, it is still fair to say that CPLEX requires much longer computational time than dynamic programming, especially for instances with larger numbers of customers. We did not use CPLEX to solve any of the other problems.

Our results match those of Ohlmann and Thomas (2007) well and match solutions by Calvo (2000) and Gendreau et al. (1998) in most cases. However, due to the cost truncation and the need to adjust for the triangular inequality using the method described in Ohlmann and Thomas (2007), some of our solution values are slightly different.

Dumas et al. (1995) report that computations for two difficult instances with $n = 100$ and $w = 60$ were not finished due to insufficient memory. However, the computational results for the instances with $n = 100$ and $w = 60$ have been suitably matched by Gendreau et al. (1998), Calvo (2000) and Ohlmann and Thomas (2007). It seems that the two challenging instances are with $n = 150$ and $w = 60$ instead of $n = 100$ and $w = 60$. Table 5 presents optimal solutions for these challenging instances and for instances with large time windows, as proposed by Dumas et al. (1995). Although instances with 100 customers and time windows of 80 units were not reported in Dumas et al.

Table 4 Comparison between DP and CPLEX for Certain Instances by Gendreau et al. (1998) without the Triangular Property

n	w	Inst	DP		CPLEX		n	w	Inst	DP		CPLEX		n	w	Inst	DP		CPLEX	
			Opt	CPU	Opt	CPU				Opt	CPU	Opt	CPU				Opt	CPU	Opt	CPU
20	120	1	267	0.17	267	0.13	20	140	1	176	0.46	176	7.11	20	160	1	241	1.59	241	18.57
		2	218	0.80	218	74.57			2	272	0.04	272	24.69			2	201	0.17	201	11.22
		3	303	0.06	303	8.49			3	236	0.13	236	32.44			3	201	0.13	201	1.64
		4	300	0.03	300	11.29			4	255	0.30	255	264.90			4	203	4.46	203	12.28
		5	240	0.51	240	0.27			5	225	0.19	225	10.27			5	245	0.39	245	73.83
		Avg	265.6	0.31	265.6	18.95			Avg	232.8	0.22	232.8	67.88			Avg	218.2	1.35	218.2	23.51
20	180	1	253	0.36	253	14.01	20	200	1	233	14.50	233	64.69	40	120	1	434	36.94	434	8h11m38s
		2	265	0.54	265	34.42			2	203	102.60	203	118.47			2	444	64.55	444	705.35
		3	271	0.41	271	71.33			3	249	5.09	249	70.68			3	357	48.75	357	2832.90
		4	201	3.23	201	31.50			4	293	3.23	293	349.54			4*	303	98.75	303	42h43m43s
		5	193	41.71	193	372.56			5	227	39.25	227	19.06			5	350	2.85	350	157.53
		Avg	236.6	9.25	236.6	104.76			Avg	241.0	32.93	241.0	124.49			Avg	377.6	50.37	377.6	10h23m23s

*: CPLEX ran on a Sun workstation with 2 processors at 900MHz and 4GB of RAM.

(1995), the data set is available in Thomas (2008). Our DP algorithm solved these instances to optimality.

Table 5 Optimal Solutions for Certain Additional Instances by Dumas et al. (1995)

n	w	Inst	With Triangle			Without Triangle			n	w	Inst	With Triangle			Without Triangle		
			Opt	$\|L\|$	CPU	Opt	$\|L\|$	CPU				Opt	$\|L\|$	CPU	Opt	$\|L\|$	CPU
150	60	1	854	65125	441.73	857	317486	965.44	100	80	1	670	69401	254.71	670	507769	765.12
		2	780	65324	163.80	782	379473	450.27			2	666	51309	165.85	667	436668	470.16
		3	793	59914	312.68	793	414329	1193.81			3	691	80167	77.35	691	306889	164.97
		4	817	61377	777.96	817	500921	2299.86			4	700	101919	179.03	700	450495	346.85
		5	840	72682	88.11	840	270962	129.38			5						
		Avg	816.8	64884	356.86	817.8	376634	1007.75			Avg						

Table 6 lists results for the instances proposed by Solomon (1987). Time windows for these instances are not uniformly distributed. We solved 19 of the 31 instances to optimality.

Table 6 Results for Instances by Solomon (1987)

Instance	n	Opt	$\|L\|$	CPU	Instance	n	Opt	$\|L\|$	CPU	Instance	n	Opt	$\|L\|$	CPU
rc201.1	19	444.54	330	0.02	rc202.1	32	771.77	230887	174.57	rc203.1	18	453.48	25525	2.54
rc201.2	25	711.54	332	0.01	rc202.2	13	304.14	10381	0.59	rc203.2	32			
rc201.3	31	790.61	825	0.02	rc202.3	28	837.72	1155	0.04	rc203.3	36			
rc201.4	25	793.63	237	0.03	rc202.4	27	793.03	148615	95.48	rc203.4	14			
rc204.1	44				rc205.1	13	343.21	539	0.03	rc206.1	3	117.85	31	0.02
rc204.2	32				rc205.2	26	755.92	4964	0.16	rc206.2	36	828.06	103318	28.83
rc204.3	23				rc205.3	34	825.06	237381	187.19	rc206.3	24	574.42	39952	5.94
rc204.4	13	343.21	539	0.05	rc205.4	27	760.47	2441	0.05	rc206.4	37	831.67	166437	78.2
rc207.1	33				rc208.1	37								
rc207.2	30				rc208.2	28								
rc207.3	32				rc208.3	35								
rc207.4	5	119.64	252	0.05										

Tables 7 and 8 present optimal solutions and comparisons for the instances proposed by Dumas et al. (1995) and Langevin et al. (1993). Our results match the solutions by Dumas et al. (1995) well in most cases. However, similar to the instances by Gendreau et al. (1998), some of our solution values were slightly smaller than those by Dumas et al. (1995) due to travel-time truncation and adjusting. These smaller solutions are identified by bold typeface. Our algorithm matches results by Langevin et al. (1993) well for those instances with 20 and 40 customers. However, computation for problems involving 60 customers remained incomplete in Langevin et al. (1993): 7 of 10 instances with time windows of 20 units were successfully finished; 8 of 10 instances with time windows of 30

Table 7 Optimal Solutions for Instances by Dumas et al. (1995)

n	w	With Triangle DP		Without Triangle DP		Without Triangle Dumas et al.		n	w	With Triangle DP		Without Triangle DP		Without Triangle Dumas et al.	
		Opt	CPU	Opt	CPU	Opt	CPU			Opt	CPU	Opt	CPU	Opt	CPU
20	20	360.6	0.02	361.2	0.02	361.2	0.02	40	20	486.4	0.02	**486.4**	0.02	486.6	0.08
	40	316.0	0.02	316.0	0.02	316.0	0.05		40	461.0	0.02	461.0	0.03	461.0	0.24
	60	309.6	0.02	309.8	0.02	309.8	0.14		60	416.4	0.15	416.4	0.16	416.4	4.37
	80	310.8	0.02	311.0	0.02	311.0	0.23		80	399.8	0.23	399.8	0.22	399.8	7.52
	100	275.2	0.04	275.2	0.04	275.2	1.27		100	377.0	6.41	377.0	13.24	377.0	31.4
60	20	581.2	0.02	581.6	0.02	581.6	0.15	80	20	676.6	0.03	676.6	0.03	676.6	0.35
	40	590.0	0.05	**590.0**	0.05	590.2	0.89		40	629.6	0.10	**629.8**	0.10	630.0	2.68
	60	559.2	0.26	560.0	0.28	560.0	6.84		60	605.8	13.06	606.4	28.62	606.4	55.32
	80	507.8	5.17	508.0	7.20	508.0	46.62		80	593.4	54.97	593.8	104.03	593.8	220.29
	100	514.8	60.46	514.8	95.55	514.8	199.84								
100	20	757.2	0.04	**757.4**	0.04	757.6	0.62	150	20	867.4	0.13	**868.2**	0.13	868.4	2.44
	40	700.4	0.37	**701.4**	0.35	701.8	7.4		40	833.2	19.85	**834.2**	31.91	834.8	115.86
	60	695.2	13.26	696.6	24.32	696.6	107.95		60	816.8	356.86	817.8	1007.75	805.0	462.97
200	20	1007.8	0.48	1009.0	0.48	1009.0	6.65	200	40	982.6	41.44	**983.8**	98.36	984.2	251.41

Table 8 Results for Instances by Langevin et al. (1993)

n	w	Inst	DP		Langevin et al.		n	w	Inst	DP		Langevin et al.	
			Opt	CPU	Opt	CPU				Opt	CPU	Opt	CPU
20	20	Avg	724.7	0.1	724.7	0.4	40	20	Avg	982.7	0.1	982.7	1.7
	30	Avg	721.5	0.1	721.5	0.7		40	Avg	951.8	0.1	951.8	7.3
60	20	1	1353.5	0.1			60	30	1	1337.0	0.1		
		2	1161.6	0.1					2	1089.5	0.1		
		3	1182.9	0.1					3	1179.0	0.1		
		4	1257.5	0.1					4	1230.0	0.1		
		5	1184.1	0.1					5	1151.6	0.1		
		6	1199.6	0.1					6	1167.9	0.1		
		7	1299.0	0.1					7	1220.1	0.1		
		8	1113.0	0.1					8	1097.6	0.1		
		9	1171.3	0.1					9	1140.6	0.1		
		10	1234.3	0.1					10	1219.2	0.1		
		Avg	1215.7	0.1	1196.4	9.6			Avg	1183.2	0.1	1180.6	32.1
60	40	1	1335.0	0.1			60	40	6	1140.2	0.1		
		2	1088.1	0.1					7	1198.9	0.1		
		3	1173.7	0.1					8	1029.4	0.1		
		4	1184.7	0.1					9	1121.4	0.1		
		5	1146.2	0.1					10	1189.6	0.1		
									Avg	1160.7	0.1	1153.9	43.4

units were finished; and 7 of 10 instances with time windows of 40 units were finished (Ohlmann and Thomas 2007). It is not surprising that our average optimal solutions for problems involving 60 customers are different from those by Langevin et al. (1993). Due to the use of different processors, memory, computer languages, and operating systems, it is difficult to compare the computational time of our DP algorithm with Langevin et al. (1993) and Dumas et al. (1995).

In summary, the bi-directional resource-bounded label correcting algorithm performs well in solving TSPTW problems. The computation is fast when the time windows are relatively narrow. For the TSPTW problem with relatively wide time windows, we were able to generate optimal solutions for certain instances that were proposed by Gendreau et al. (1998) and Solomon (1987). However, our algorithm appeared incapable of solving those instances with much larger number of customers or with much wider time windows. The program quits due to insufficient memory or is stopped manually due to long computational times. We note that our dynamic programming algorithm is more sensitive to wide time windows than to large numbers of customers. For example, we solved 4 out of the 5 instances with 100 customers and time windows of 80 units (see Table 5). However, we were unable to solve any instances involving 80 customers and time windows of 100 units.

Even though we do not focus on comparing our exact algorithm to heuristic algorithms in the literature, certain heuristic algorithms have already generated optimal solutions for a large number of instances, especially those by Ohlmann and Thomas (2007), Gendreau et al. (1998) and Calvo (2000).

4. Conclusions and Future Research

In this paper, we design a bi-directional resource-bounded label correcting algorithm to solve the traveling salesman problem with time windows (TSPTW). The bi-directional dynamic programming starts label extension and dominance simultaneously from both the starting and terminating depots. The search space is significantly reduced since each direction can consume approximately half of the available resources. The labels generated in the two directions are then joined if the relevant feasibility conditions are met. If the triangular inequality holds, the dominance rules can be further strengthened and the corresponding computational time is substantially reduced.

Our bi-directional resource-bounded dynamic programming optimally solved certain benchmark problems with wide time windows, which to our knowledge have not been solved elsewhere. These problems are as follows.

1: problems proposed by Gendreau et al. (1998)
all the instances with 20 customers and time windows of 120, 140, 160, 180, and 200 units; most instances with 40 customers with time windows of 120, 140 and 160 units; and certain instances with 60 customers with time windows of 120 and 140 units.

2: problems proposed by Dumas et al. (1995)
two difficult instances with 150 customers and time windows of 60 units; and 4 out of 5 instances with 100 customers and time windows of 80 units.

3: problems proposed by Solomon (1987)
19 out of 31 instances with varying numbers of customers and time windows.

4: problems proposed by Langevin et al. (1993)
all the instances involving 60 customers are solved to optimality.

Future research may investigate techniques to further reduce the number of labels, such as the state space relaxation technique (Christofides et al. 1981, Righini and Salani 2008). Other TSP variations, such as the TSP with precedence constraints, can also be studied.

Acknowledgments

The author would like to thank Dr. Barret Thomas for providing the benchmark data and for offering advices regarding how to calculate travel times.

References

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

Beasley, J. E., N. Christofides. 1989. An algorithm for the resource constrained shortest-path problem. *Networks* **19**(4) 379–394.

Boland, N., J. Dethridge, I. Dumitrescu. 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters* **34** 58–68.

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

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

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 Science* **42**(3) 387–404.

Desrochers, M., F. Soumis. 1988. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR* **26**(3) 191–212.

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

Feillet, D., P. Dejax, M. Gendreau, C. Gueguen. 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks* **44**(3) 216–229.

Focacci, F., A. Lodi, M. Milano. 2002. A hybrid exact algorithm for the TSPTW. *INFORMS Journal on Computing* **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. *Operations Research* **46**(3) 330–335.

Kohl, N., J. Desrosiers, O. B. G. Madsen, M. M. Solomon, F. Soumis. 1999. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science* **33**(1) 101–116.

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

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

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 Science* **32**(1) 12–29.

Pesant, G., M. Gendreau, J. Y. Potvin, J. M. Rousseau. 1999. On the flexibility of constraint programming models: From single to multiple time windows for the traveling salesman problem. *European Journal of Operational Research* **117**(2) 253–263.

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

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

Righini, G., M. Salani. 2008. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks* **51**(3) 155–170.

Savelsbergh, M. W. P. 1985. Local search in routing problems with time windows. *Annals of Operations Research* **4**(1) 285–305.

Solomon, M. M. 1987. Algorithms for the vehicle-routing and scheduling problems with time window constraints. *Operations Research* **35**(2) 254–265.

Thomas, B. W. 2008. Traveling salesman problem with time windows - benchmark data sets. http://myweb.uiowa.edu/bthoa/TSPTWBenchmarkDataSets.htm.