A Computational Study of Bi-directional Dynamic Programming for the Traveling Salesman Problem with Time Windows

Jing-Quan Li

California PATH, University of California, Berkeley, Richmond, CA 94804,
jingquan@path.berkeley.edu

Abstract

This paper presents computational studies for the traveling salesman problem with time windows by applying a bi-directional resource-bounded label correcting algorithm. 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. We conduct extensive computational experiments on six benchmark data sets available in the literature. Optimal solutions are reported for a number of the instances for which no optimal solutions have been previously reported. The approach developed is shown competitive for the traveling salesman problem with time windows if compared with state-of-art methods in the literature.

Key words: Dynamic Programming, Time Windows, Bi-directional Resource-bounded 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 [14]. The TSPTW also arises as a sub-problem in the vehicle routing context if a cluster-first route-second approach is employed [14]. 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 [15; 8].

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 t_{ij} and c_{ij} be the time and cost of traveling from customer i to customer j, respectively. The time window of customer i is $[a_i, b_i]$. The service time of customer i is W_i. Without the loss of generality, W_i can be included into the arc time t_{ij}, and from now on, we assume $W_i = 0$. The objective is to minimize the total travel cost. 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 [25] 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 [3], the branch-and-cut algorithm [2], the two-commodity flow formulation with the branch-and-bound scheme [16], the dynamic programming with label reduction techniques [11], and the variations of constraint programming [19; 13]. Other related approaches include the dynamic programming for certain restricted TSP problems [4] and TSP with precedence constraints [17]. Heuristics methods include the generalized insertion heuristic with local re-optimization [14], the constraint programming [20], the construction heuristic with local search improvements [7] and the compressed-annealing heuristic [18].

As shown in Dumas et al. [11], 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. [11] 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. [14].

Dynamic programming has also been widely used to solve the resource-constrained shortest path problem (for an example, see [9]). Recently, bi-directional resource-bounded dynamic programming approaches have been proposed by Righini and Salani [22; 23] and Boland et al. [6] to solve the elementary shortest path problem with resource constraints, a NP-hard problem in the strong sense [10]. Righini and Salani [22; 23] and Boland et al. [6] 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 [22; 23], we designed a bi-directional resource-bounded dynamic programming algo-

rithm to solve the TSPTW problems, with the focus on computational studies. Extensive computational experiments are conducted based on six benchmark data sets. Comparisons with related algorithms on the literature are also provided.

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 [5], an additional resource vector, 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 by the path corresponding to 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. $s^{'} = \max\{s + t_{ij}, a_j\}$ since the salesman has to wait if the arrival time to node j is before the time window. $s + t_{ij}$ is the *earliest arrival time* to node j from node i. $V_k^{'} = V_k$ if $k \neq j$, and $V_j^{'} = V_i + 1$. The extension is feasible if (i) j has not previously been visited ($V_j = 0$) and (ii) $s^{'}$ is within the time window($a_j \leq s^{'} \leq b_j$). 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. [11] 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 [22], we can determine the maximum feasible arrival time to the terminating depot (node $n+1$), say $T = \max_{i \in N}\{b_i + t_{i,n+1}\}$. 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 $s^{'} = \min\{s - t_{ji}, b_j\}$ since service cannot be started after b_j. $s - t_{ji}$ indicates the *latest departure time* at node j. $V_k^{'} = V_k$ if $k \neq j$, and $V_j^{'} = V_i + 1$. The backward extension operation is feasible if (i) j has not previously been visited ($V_j = 0$), (ii) $s^{'}$ is no later than the time window ($s^{'} \geq a_j$), 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 [22] 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 ($c_{ij} + c_{jk} > c_{ik}$ and $t_{ij} + t_{jk} > t_{ik}$ for customers i, j, and k, where c_{ij} and t_{ij} represents the cost and time traveling from i to j, respectively) 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 [12].

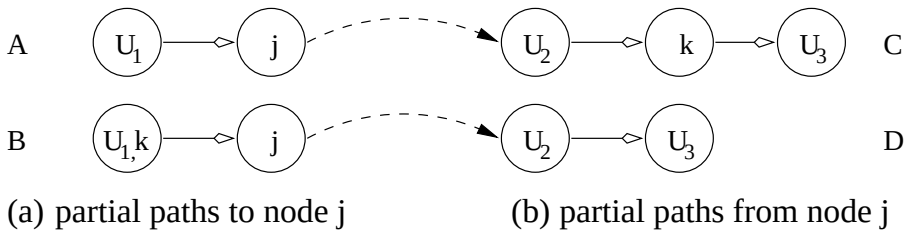

Figure 1: An Example of the Triangle Inequality

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. Note that due to $t_{U_2k} + t_{kU_3} > t_{U_2U_3}$, traveling from U_2 to U_3 is still feasible after k is removed. 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 [22] 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 [22] 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. Joining labels of different nodes needs to scan the network, which may be time consuming in some instances. 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. Figure 2.(a) presents an example of label joining at the same node, where short line segments indicate nodes with time windows. There are six nodes, and nodes 0 and 5 are the starting and terminating depot respectively. The starting time from the depot is 0; and the ending time to the depot is the upper bound, T. Let T_b be the boundary of the time resource. In the case of the forward extension, the earliest arrival time needs to be less than or equal to T_b, whereas in the case of the backward extension, the latest departure time needs to be more than or equal to T_b, as can be seen in Figure 2.(a). The backward extension from node 3 to node 2 is disallowed since the latest departure time is less than T_b.

However, such way of joining a forward and a backward label from the same node may result in a potential problem. The resource bounding may lead to three potential situations, as shown in Figure 2.(b-d). In Figure 2.(b), T_b allows the backward extension from node 3 to 2 and disallows the forward extension from node 2 to node 3. Label joining can be done at node 2. In Figure 2.(c), T_b allows the forward extension from node 2 to node 3 and disallows the backward extension from node 3 to 2. Label joining can be done at node 3. In Figure 2.(d), T_b disallows the forward extension from node 2 to node 3 as well as the backward extension from node 3 to 2. Label joining cannot be done at either of nodes 2 and 3. Therefore, the optimal solution can never be reached. This potential problem is not present in the approach by Righini and Salani [22] 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 backward label at node 3 can be extended to reach node 2. In such case 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 in the forward extension from (V, s, i) to node j, the extension is allowed only if $s + t_{ij} \leq T_b$; and in the backward extension from (V, s, i) to node j, the extension is allowed only if $s - t_{ji} \geq T_b - G$. T_b is often set to $T/2$ but not limited to $T/2$.

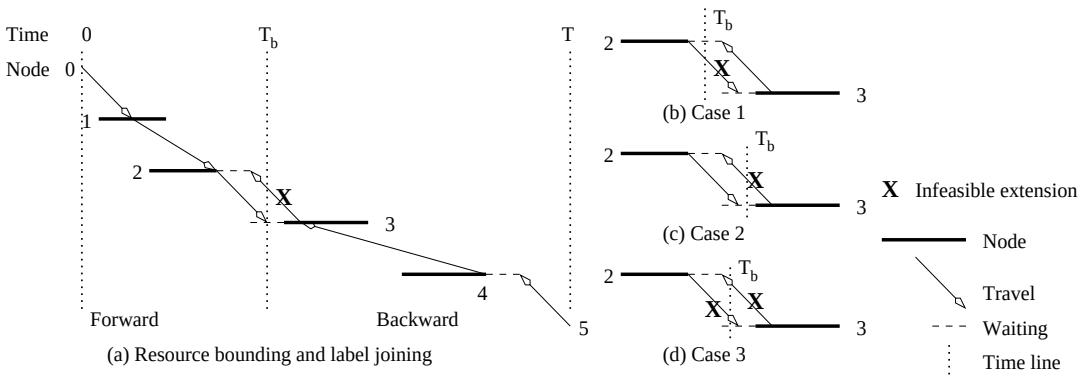

Figure 2: Label Joining, Resource Bounding and a Potential Problem

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 $T_b = 10/2 - 2$ (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.

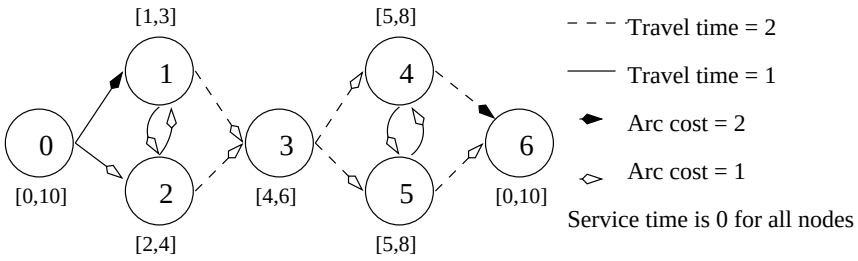

Figure 3: An example of bi-directional resource-bounded DP

Table 1: Labels in the Dynamic Programming Example

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

The pseudo-code of the DP algorithm is described in the appendix.

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

Six benchmark data sets are widely used in the literature.

- Problems proposed by Dumas et al. [11]

  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. The service time is zero. The arc cost is same as the arc distance.

- Problems proposed by Gendreau et al. [14]

  These instances are similar to ones by Dumas et al. [11], but time windows are relatively large.

- Problems proposed by Langevin et al. [16]

  Each problem with the same number of customers and time window widths has 10 instances. A travel-time matrix with fractional values is available. The arc cost is same as the arc time.

- Problems proposed by Potvin and Bengio [21] based on the routing problems by Solomon [26]

  These problems are generated as individual routes from Solomon's RC2 instances [26] in the vehicle routing problem with time windows by the tabu search heuristic of Potvin and Bengio [21]. 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. Time windows for these instances are not uniformly distributed. The arc cost is the arc traversal time plus the service time at the starting node.

- Problems proposed by Pesant et al. [19] based on the routing problems by Solomon [26]

  Pesant et al. [19] also obtain the individual routes from Solomon's RC2 instances Solomon [26]. However, the individual routes are produced by Rochat and Taillard [24] and Taillard et al. [27]. Therefore, the TSPTW instances by the methods of Pesant et al. [19] are different from ones by Potvin and Bengio [21]. The arc traversal time is calculated as the fractional Euclidean distance. The arc cost is same as the arc traversal time. The service time is not included into the arc cost.

- Asymmetric TSPTW Instances by Ascheuer et al. [2]

  Ascheuer et al. [2] proposed real-world asymmetric TSPTW instances that come from an industry project with the objective to minimize the unloaded travel time of a stacker crane. An arc time matrix with fractional values is available. The arc cost is same as the arc time.

Table 3 gives a summary of the date sets and download locations. Most instances by Dumas et al. [11] and Langevin et al. [16] have relatively narrow time windows and have already been solved to optimality. Instances proposed by Gendreau et al. [14] exhibit wide time windows; to the best of our knowledge, no optimal solutions have been reported for the instances by Gendreau et al. [14] and Potvin and Bengio [21]. A large number of instances by Pesant et al. [19] and Ascheuer et al. [2] have been solved to optimality [19; 2; 13].

Table 3: Benchmark Instances

Instance	Arc Distance	Available at
Langevin et al. [16]	available as fractional values	http://myweb.uiowa.edu/bthoa/Data Sets.html
Dumas et al. [11]	calculate using the coordinates of nodes	http://myweb.uiowa.edu/bthoa/Data Sets.html
Gendreau et al. [14]	calculate using the coordinates of nodes	http://myweb.uiowa.edu/bthoa/Data Sets.html
Potvin and Bengio [21]	calculate using the coordinates of nodes	http://myweb.uiowa.edu/bthoa/Data Sets.html
Pesant et al. [19]	calculate using the coordinates of nodes	http://www.crt.umontreal.ca/∼ quosseca/detail_publication_eng.php?id= 56
Ascheuer et al. [2]	available as integral values	http://elib.zib.de/pub/mp-testdata/tsp/atsptw/text.en.html

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. [14] 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.

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.

3.2 Computational Results

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

3.2.1 Instances by Dumas et al. [11]

The arc traversal distance is not directly available in the instances by Dumas et al. [11]. When the underlying network is generated, Dumas et al. [11] truncate the fractional travel times into integral data, and no decimal places are kept. However, Pesant et al. [19] point out that an optimal solution obtained with truncated distances may be even infeasible if a larger precision is used. Pesant et al. [19] then proposed to truncate the fractional travel times after four decimal places. We follow this way to truncate the travel distances and then generate the underlying network.

Table 4: Instances by Dumas et al. [11] with the Precision of Four Decimal Places

n	w	Pesant et al.			DP			n	w	Pesant et al.			DP		
		F	Opt	CPU	F	Opt	CPU			F	Opt	CPU	F	Opt	CPU
20	20	4	363	1.8	4	362.9980	0.0	40	20	3	497.2	10.3	3	497.1583	0.0
	40	5	328.2	3.5	5	328.1862	0.0		40	5	477.6	1111.1	5	477.5615	0.0
	60	5	317.7	14.4	5	317.7246	0.0		60	5	432.2	31386.3	5	432.2344	0.1
	80	5	329.2	125.9	5	329.1794	0.0		80		-	-	5	414.9140	0.2
	100	5	281.5	102.2	5	281.5441	0.0		100		-	-	5	392.3054	4.2
60	20	3	588.2	134.5	3	588.1924	0.0	80	20	2 to 3	713.3	511.9	3	692.9180	0.0
	40	2 to 5	593.2	12189.7	5	615.3176	0.1		40	-	-	-	5	661.2398	0.1
	60	-	-	-	5	582.6412	0.2		60	-	-	-	4	628.5021	24.6
	80	-	-	-	5	530.3669	5.3		80	-	-	-	5	622.0359	119.4
	100	-	-	-	5	538.4324	89.8								
100	20	1	775.2	18003.5	2	763.2790	0.1	150	20	-	-	-	3	925.6249	0.3
	40	-	-	-	5	738.9197	0.5		40	-	-	-	4	861.7535	37.2
	60	-	-	-	5	735.1255	22.2		60	-	-	-	5	873.4769	1385.2
200	20	-	-	-	1	1049.2000	1.4	200	40	-	-	-	4	1065.0065	55.2

infeasible instances: n20/w20 (inst 3); n40/w20 (inst 1 and 5); n60/w20 (inst 2 and 4); n80/w20 (inst 3 and 5); n80/w60 (inst 3); n100/w20 (inst 3, 4 and 5); n150/w20 (inst 2); n150/w40 (inst 2); n200/w20 (inst 1, 3, 4 and 5); n200/w40 (inst 2);

Table 4 presents our solutions and comparisons with ones by Pesant et al. [19]. "-" indicates that no data is available; and F indicates the number of feasible instances. We either solve an instance to optimality or show that there is no a feasible solution for the instance.

Our results confirm the finding by Pesant et al. [19] that some optimal solutions based on the truncated distances are infeasible with the precision of four decimal places, especially for instances with large n and small w. For example, only two instances with $n = 100$ and $w = 20$ are feasible. Our results match the solutions by Pesant et al. [19] well in most cases, say for instances with $n = 20$ and $n = 40$. Some infeasible instances in Pesant et al. [19] may also be caused by exceeding the time limit. Our results do not match well for such instances, for example ones with $n = 60/w = 40$ and $n = 80/w = 20$. Due to the use of different processors, memory, computer languages, and operating systems, it is difficult to compare the computational time of different algorithms. However, we feel the DP algorithm is competitive for the instances by Dumas et al. [11].

Dumas et al. [11] 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 [14; 7; 18]. 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. [11]. Although instances with 100 customers and time windows of 80 units were not reported in Dumas et al. [11], the data set is available in [28]. The DP algorithm solved most of these instances to optimality.

Table 5: Certain Additional Instances by Dumas et al. [11] with Four Decimal Places

n	w	Inst	Opt	$\|L\|$	CPU	n	w	Inst	Opt	$\|L\|$	CPU
150	60	1	918.3608	317692	1380.96	100	80	1	704.9239	362219	361.01
		2	828.7122	215651	105.65			2	710.3789	395202	380.48
		3	851.0369	358491	1040.09			3	735.6319	272390	164.10
		4	871.0867	603031	4306.93			4	740.9760	358952	202.15
		5	898.1880	217548	92.29			5	-	-	-

3.2.2 Instances by Gendreau et al. [14]

Certain studies [14; 18] 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 [18] 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 (please see [28] for further detail). As mentioned by [28], the triangular inequality is not guaranteed to be maintained by this procedure, an outcome that is confirmed in our experiments. We first follow this strategy to generate the underlying network. Note that the fractional travel times are truncated and no decimal places are kept. Then, we generate the underlying network with the triangular property by repeatedly calling the above procedure until no further violation is identified. Comparisons between the networks with the triangular property and without the triangular property are also provided.

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 apply the method of [28] to truncate the fractional travel times to integer values. Our optimal solutions match those of CPLEX very well (see columns 4, 6, 11, 13, 17 and 19 in Table 6). 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.

Table 6: Comparison between DP and CPLEX for Certain Instances by Gendreau et al. [14]

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	0.23	267	0.13	20	140	1	176	0.63	176	7.11	20	160	1	241	1.57	241	18.57
		2	218	0.51	218	74.57			2	272	0.13	272	24.69			2	201	0.23	201	11.22
		3	303	0.11	303	8.49			3	236	0.19	236	32.44			3	201	0.19	201	1.64
		4	300	0.1	300	11.29			4	255	0.37	255	264.90			4	203	4.47	203	12.28
		5	240	0.48	240	0.27			5	225	0.23	225	10.27			5	245	0.37	245	73.83
		Avg	265.6	0.29	265.6	18.95			Avg	232.8	0.31	232.8	67.88			Avg	218.2	1.37	218.2	23.51
20	180	1	253	0.43	253	14.01	20	200	1	233	12.64	233	64.69	40	120	1	434	46.04	434	29498.11
		2	265	0.44	265	34.42			2	203	65.76	203	118.47			2	444	60.09	444	705.35
		3	271	0.41	271	71.33			3	249	4.85	249	70.68			3	357	47.84	357	2832.90
		4	201	3.23	201	31.50			4	293	2.7	293	349.54			4	303	52.76	303	153823.54
		5	193	32.6	193	372.56			5	227	28.97	227	19.06			5	350	2.5	350	157.53
		Avg	236.6	7.42	236.6	104.76			Avg	241.0	22.98	241.0	124.49			Avg	377.6	41.85	377.6	37403.49

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

Table 7 lists computational results of certain instances by Gendreau et al. [14]. 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 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.

Our results match those of [18] well and match solutions by [7] and [14] in most cases. However, due to the need to adjust for the triangular inequality using the method described in [18], some of our solution values are slightly different from [7] and [14].

The DP 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 the DP algorithm is more sensitive to wide time windows

Table 7: Certain Instances by Gendreau et al. [14] without Keeping Decimal Places

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	2641	0.25	267	7058	0.23	20	140	1	176	6022	0.51	176	13777	0.63
		2	218	4856	0.35	218	10114	0.51			2	272	1835	0.1	272	2782	0.13
		3	303	987	0.1	303	2732	0.11			3	236	3144	0.18	236	5787	0.19
		4	300	1086	0.1	300	1526	0.1			4	255	5475	0.38	255	10580	0.37
		5	240	3638	0.31	240	9250	0.48			5	225	2196	0.17	225	5889	0.23
		Avg	265.6	2642	0.22	265.6	6136	0.29			Avg	232.8	3734	0.27	232.8	7763	0.31
20	160	1	241	9042	1.68	241	24168	1.57	20	180	1	253	6715	0.48	253	11858	0.43
		2	201	2938	0.19	201	6964	0.23			2	265	4253	0.35	265	9908	0.44
		3	201	2585	0.19	201	6089	0.19			3	271	7106	0.49	271	11392	0.41
		4	203	9721	3.45	203	46084	4.47			4	201	16762	3.24	201	41008	3.23
		5	245	5397	0.35	245	9808	0.37			5	193	43996	27.22	193	136001	32.6
		Avg	218.2	5937	1.17	218.2	18623	1.37			Avg	236.6	15766	6.36	236.6	42033	7.42
20	200	1	233	28466	12	233	78634	12.64	40	120	1	434	17665	15.7	434	84092	46.04
		2	203	87301	55.78	203	187077	65.76			2	444	54822	51.57	444	183323	60.09
		3	249	10767	4.56	249	48346	4.85			3	357	39164	39.17	357	173361	47.84
		4	293	18548	3.09	293	37453	2.7			4	303	23773	28.68	303	137660	52.76
		5	227	62449	26.43	227	126943	28.97			5	350	9536	2.45	350	33075	2.5
		Avg	241.0	41506	20.37	241.0	95691	22.98			Avg	377.6	28992	27.51	377.6	122302	41.85
40	140	1	328	58691	42.92	328	208718	68.96	40	160	1	348	104615	117.30	348	359288	196.12
		2	383	86595	83.23	383	300364	160.01			2	337	213833	1331.49	337	879802	3727.44
		3	398	18376	5.56	398	58002	5.43			3	346	85982	474.49	346	616596	1456.60
		4	342	140017	230.5	342	452003	403.15			4	-	-	-	-	-	-
		5	371	221795	1288.17	371	924574	3393.74			5	315	332075	9369.08	315	1883813	21653.18
		Avg	364.4	105095	330.08	364.4	388732	806.26									
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									

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.

Pesant et al. [19] show that truncating the fractional travel times into integral values has a significant impact on the instances by Dumas et al. [11]. We investigate the impacts on the instances by Gendreau et al. [14]. Table 8 presents the results when the fractional travel times are simply truncated and the precision of four decimal places is used. First, we can clearly see that the objective value with four decimal places is higher than the objective value without keeping decimal places, which confirms the finding by Pesant et al. [19]. Many instances by Dumas et al. [11] become infeasible when the precision of four decimal places is used. However, this is not the case for the instances by Gendreau et al. [14]; we either solve a problem to optimality or the computation is terminated due to insufficient memory. We believe the relatively wide time window of the instances by Gendreau et al. [14] makes the problem more feasible even if the fractional time is simply truncated. In addition, the computational time of the DP algorithm seems not to suffer from the larger precision. The computational time for the both situations is on the same order. In some cases, for example $n = 40$ and $w = 140$, the computational time with the larger precision is even smaller.

Table 8: Impact of a Larger Precision of Decimal Places

n	w	Inst	Truncating			Four Decimal			n	w	Inst	Truncating			Four Decimal		
			Opt	$\|L\|$	CPU	Opt	$\|L\|$	CPU				Opt	$\|L\|$	CPU	Opt	$\|L\|$	CPU
20	120	1	267	7058	0.23	280.8091	5794	0.21	20	140	1	176	13777	0.63	185.0359	15184	0.76
		2	218	10114	0.51	226.0163	10100	0.54			2	272	2782	0.13	280.3943	2382	0.11
		3	303	2732	0.11	310.3447	2060	0.11			3	236	5787	0.19	242.0557	5946	0.23
		4	300	1526	0.10	309.4237	1377	0.12			4	255	10580	0.37	258.6243	8363	0.28
		5	240	9250	0.48	246.6782	10840	0.72			5	225	5889	0.23	228.5938	5532	0.22
		Avg	265.6	6136	0.29	274.6544	6034	0.34			Avg	232.8	7763	0.31	238.9408	7481	0.32
20	160	1	241	24168	1.57	250.8983	24349	1.92	20	180	1	253	11858	0.43	260.5546	11394	0.41
		2	201	6964	0.23	207.8567	6068	0.22			2	265	9908	0.44	274.0112	9214	0.42
		3	201	6089	0.19	206.7017	5620	0.20			3	271	11392	0.41	278.9672	9780	0.37
		4	203	46084	4.47	209.2817	40674	3.67			4	201	41008	3.23	218.3118	41753	3.61
		5	245	9808	0.37	253.0218	10036	0.41			5	193	136001	32.60	199.1145	140977	39.92
		Avg	218.2	18623	1.37	225.5520	17349	1.28			Avg	236.6	42033	7.42	246.1919	42624	8.95
20	200	1	233	78634	12.64	239.8267	74723	12.55	40	120	1	434	84092	46.04	451.3430	88454	51.43
		2	203	187077	65.76	208.3406	186402	72.44			2	444	183323	60.09	463.2439	145610	41.16
		3	249	48346	4.85	257.1252	52382	6.52			3	357	173361	47.84	378.7716	158932	44.14
		4	293	37453	2.70	297.9340	36033	2.64			4	303	137660	52.76	316.6826	160930	112.11
		5	227	126943	28.97	231.6644	120921	28.02			5	350	33075	2.50	363.3558	27478	2.05
		Avg	241.0	95691	22.98	246.9782	94092	24.43			Avg	377.6	122302	41.85	394.6794	116281	50.18
40	140	1	328	208718	68.96	359.1646	207338	80.35									
		2	383	300364	160.01	399.2972	311241	199.02									
		3	398	58002	5.43	414.2916	52837	5.22									
		4	342	452003	403.15	354.7020	364788	243.95									
		5	371	924574	3393.74	386.4636	809616	2896.71									
		Avg	364.4	388732	806.26	382.7838	349164	685.05									

3.2.3 Instances by Langevin et al. [16]

Table 9: Instances by Langevin et al. [16]

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

Table 9 presents optimal solutions and comparisons for the instances proposed by Langevin et al. [16], where arc traversal distances is directly available as fractional values. We use float variables to identify the arc travel time in our implementation. The DP algorithm matches results by Langevin et al. [16] well for those instances with 20 and 40 customers. However, computation for problems involving 60 customers remained incomplete in Langevin et al. [16]: 7 of 10 instances with time windows of 20 units were successfully finished; 8 of 10 instances with time windows of 30 units were finished; and 7 of 10 instances with time windows of 40 units were finished [18]. It is not surprising that our average optimal solutions for problems involving 60 customers are different from those by Langevin et al. [16].

3.2.4 Instances by Potvin and Bengio [21]

Table 10 lists results for the TSPTW instances by Potvin and Bengio [21]. A precision of four decimal places is used for truncating fractional travel times into integral values. To the best of our knowledge, there were no optimal solutions reported for these instances before. We obtained optimal solutions for 19 of the 31 instances, and the results match those of [14], [7] and [18] well.

Table 10: Instances by Potvin and Bengio [21]

Inst	n	Opt	$\|L\|$	CPU	Inst	n	Opt	$\|L\|$	CPU	Inst	n	Opt	$\|L\|$	CPU
rc201.1	19	444.5418	327	0.09	rc202.1	32	771.7747	231000	164.57	rc203.1	18	453.4812	25103	2.44
rc201.2	25	711.5364	325	0.10	rc202.2	13	304.1413	10509	0.64	rc203.2	32	-	-	-
rc201.3	31	790.6058	810	0.11	rc202.3	28	837.7177	1070	0.09	rc203.3	36	-	-	-
rc201.4	25	793.6339	237	0.08	rc202.4	27	793.0283	151056	94.89	rc203.4	14	3142887	46638	11.12
rc204.1	44	-	-	-	rc205.1	13	343.2089	600	0.09	rc206.1	3	117.8478	31	0.09
rc204.2	32	-	-	-	rc205.2	26	755.9243	4814	0.21	rc206.2	36	828.0577	93347	20.59
rc204.3	23	-	-	-	rc205.3	34	825.0572	242628	197.70	rc206.3	24	574.4173	38793	5.52
rc204.4	13	343.2089	600	0.10	rc205.4	27	760.4693	2441	0.12	rc206.4	37	831.6686	156776	69.60
rc207.1	33	-	-	-	rc208.1	37	-	-	-	-	-	-	-	-
rc207.2	30	-	-	-	rc208.2	28	-	-	-	-	-	-	-	-
rc207.3	32	-	-	-	rc208.3	35	-	-	-	-	-	-	-	-
rc207.4	5	119.6384	252	0.10										

3.2.5 Instances by Pesant et al. [19]

Table 11 shows computational results for the TSPTW instances by Pesant et al. [19]. A precision of four decimal places is used for truncating fractional travel times into integral values. A large number of TSPTW instances by Pesant et al. [19] have seen solved to optimality by the constraint programming [19] and hybrid constraint and integer programming [13]. Table 11 lists our computational results and comparisons with ones by Pesant et al. [19] and Focacci et al. [13], where Val indicates the objective value, * indicates an optimal solution is found, and - shows that no data is available. Focacci et al. [13] use both assignment problem (AP) and Lagrangian relaxation as bounding techniques.

The Lagrangian-bound version of Focacci et al. [13] solve most instances to optimality: 23 over 27 instances. Pesant et al. [19] obtain optimal solutions for 13 instances. The DP algorithm solved 18 instances to optimality, five less than Focacci et al. [13]. The DP solves some instances very quickly. For example,

Table 11: Instances by Pesant et al. [19]

Inst	n	$\|A\|$	Pesant et al.		Focacci et al. AP		Focacci et al. Lagrangian		DP		
			Val	CPU	Val	CPU	Val	CPU	Val	L	CPU
rc201.0	25	191	378.62*	51.1	378.62*	0.2	378.62*	0.2	378.62*	547	0.1
rc201.1	28	238	374.7*	311.0	374.7*	1.8	374.7*	4.7	374.7*	485	0.1
rc201.2	28	231	427.65*	261.8	427.65*	0.8	427.65*	0.4	427.65*	387	0.1
rc201.3	19	128	232.54*	6.5	232.54*	0.1	232.54*	0.1	232.54*	344	0.1
rc202.0	25	446	246.22*	4136.0	246.22*	8.5	246.22*	1.0	246.22*	175714	63.0
rc202.1	22	301	206.53*	1803.9	206.53*	3.8	206.53*	10.4	206.53*	20386	1.1
rc202.2	27	398	341.77*	696.2	341.77*	3.1	341.77*	5.0	341.77*	11520	0.4
rc202.3	26	438	367.85*	11107.7	367.85*	33.1	367.85*	105.5	367.85*	48976	5.1
rc203.0	35	1084	384.8	-	396.2	-	388.71	-	-	-	-
rc203.1	37	1201	357.3	-	399.9	-	377.31	-	-	-	-
rc203.2	28	561	337.6	-	337.46*	166.4	337.46*	94.3	337.46*	369604	392.7
rc204.0	32	937	221.5	-	221.45*	1295.1	221.45*	352.8	-	-	-
rc204.1	28	740	206.4	-	205.37*	993.5	205.37*	3.4	-	-	-
rc204.2	40	1472	379	-	430.32	-	420.15	-	-	-	-
rc205.0	26	387	251.65*	652.4	251.65*	9.2	251.65*	7.9	251.65*	25987	1.9
rc205.1	22	232	271.22*	128.1	271.22*	0.2	271.22*	0.3	271.22*	9426	0.4
rc205.2	28	384	436.64	-	434.69*	1289.1	434.69*	1409.4	434.69*	9616	0.3
rc205.3	24	306	361.24*	7026.2	361.24*	4.7	361.24*	5.1	361.24*	2717	0.1
rc206.0	35	697	495.8	-	485.23	-	485.23*	338.1	485.23*	29629	1.6
rc206.1	33	696	334.8	-	334.73*	48.9	334.73*	22.9	334.73*	210574	42.5
rc206.2	32	680	335.37*	67543.1	335.37*	170.4	335.37*	24.1	335.37*	335440	189.5
rc207.0	37	1076	436.69	-	436.69*	572.0	436.69*	991.0	436.69*	2473988	18691.8
rc207.1	33	867	396.36	-	396.36	-	396.36*	321.7	396.36*	1913260	10864.5
rc207.2	30	770	246.41*	6790.6	246.41*	299.8	246.41*	15.1	-	-	-
rc208.0	44	1980	381.1	-	-	-	-	-	-	-	-
rc208.1	27	756	239.1	-	239.04*	242.6	239.04*	34.2	-	-	-
rc208.2	29	870	214	-	213.92*	13.9	213.92*	1.4	-	-	-

it takes only 0.3 second to solve rc205.2 and 1.6 seconds to solve rc206.0. However, 10864.5 seconds are spent to solve rc207.2, whereas it takes Lagrangian-bound version of Focacci et al. [13] only 321.7 seconds to obtain an optimal solution. The computational time of the DP algorithm is relatively sensitive for the Solomon instances by Pesant et al. [19] and Potvin and Bengio [21], which may be caused by non-uniformly distributed time windows.

3.2.6 Asymmetric TSPTW Instances by Ascheuer et al. [2]

We compare the results by the branch-and-cut algorithm [2], AP-bound and Lagrangian-bound version [13], and the DP algorithm. The AP-bound version [13] performs better than Lagrangian-bound version for the instances by Ascheuer et al. [2]. Both of the branch-and-cut algorithm of Ascheuer et al. [2] and AP-bound version of Focacci et al. [13] solve 31 instances to optimality. Focacci et al. [13] did not solve rbg125a to optimality but found an optimal solution for rbg042a. Ascheuer et al. [2] is just opposite: rbg125a is solved to optimality, while rbg042a is not. Among 31 instances that have been solved to optimality by Ascheuer et al. [2], we found optimal solutions for 26 instances. Among 31 instances that have been optimally solved by Focacci et al. [13], we also solve 26 instances to optimality. Five instances we fail to obtain solutions are: rbg021.7, rbg021.8, rbg021.9, rbg035a.2, and rgb050a. However, we obtain optimal solutions for nine additional instances with a large number of customers. These instances are rbg041a, rbg086a, rbg092a, rbg132, rbg152, rbg172a, rbg193, rbg201a, and rbg233. Note that rbg233 is the instance that has the largest number of customers. In total, we solved 36 instances to optimality.

Table 12: Instances by Ascheuer et al. [2]

Instance	n	$\|A\|$	Ascheuer et al.		Focacci et al. AP		Focacci et al. Lagrange		DP		
			Val	CPU	Val	CPU	Val	CPU	Val	$\|L\|$	CPU
rbg010a	12	54	149*	0.1	149*	0.0	149*	0.1	149*	143	0.1
rbg016a	18	79	179*	0.2	179*	0.1	179*	0.1	179*	230	0.1
rbg016b	18	167	142*	8.8	142*	0.1	142*	0.2	142*	3190	0.1
rbg017.2	17	200	107*	0	107*	0.0	107*	0.1	107*	69388	19.9
rbg017	17	122	148*	0.8	148*	0.1	148*	0.1	148*	1288	0.1
rbg017a	19	176	146*	0.1	146*	0.1	146*	0.1	146*	65999	10.3
rbg019a	21	71	217*	0	217*	0.0	217*	0.1	217*	71	0.1
rbg019b	21	211	182*	54.6	182*	0.2	182*	0.2	182*	6691	0.2
rbg019c	21	229	190*	8.7	190*	0.3	190*	0.7	190*	123503	35.3
rbg019d	21	156	344*	0.7	344*	0.0	344*	0.1	344*	360	0.1
rbg020a	22	95	210*	0.2	210*	0.0	210*	0.1	210*	65469	10.5
rbg021.2	21	237	182*	0.2	182*	0.2	182*	0.3	182*	160968	48.0
rbg021.3	21	256	182*	27.1	182*	0.4	182*	0.5	182*	217121	101.2
rbg021.4	21	264	179*	5.8	179*	0.3	179*	0.4	179*	450251	692.8
rbg021.5	21	268	169*	6.6	169*	0.2	169*	0.3	169*	603226	1320.5
rbg021.6	21	358	134*	1.3	134*	0.9	134*	0.7	134*	1288704	7876.4
rbg021.7	21	375	133*	4.3	133*	0.6	133*	1.0	-	-	-
rbg021.8	21	380	132*	17.4	132*	0.8	132*	0.6	-	-	-
rbg021.9	21	380	132*	26.1	132*	0.8	132*	0.9	-	-	-
rbg021	21	229	190*	8.5	190*	0.3	190*	0.7	190*	123503	35.5
rbg027a	29	487	268*	2.2	268*	0.2	268*	0.3	268*	13646	0.7
rbg031a	33	388	328*	1.7	328*	12.9	328*	2.7	328*	4025	0.1
rbg033a	35	421	433*	1.8	433*	1.0	433*	-	433*	7332	0.2
rbg034a	36	535	403*	1	403*	55.2	403*	777.1	403*	19498	1.0
rbg035a.2	37	940	166*	1.8	166*	36.8	166*	38.4	-	-	-
rbg035a	37	477	254*	64.8	254*	3.5	254*	11.9	254*	6307	0.1
rbg038a	40	486	466*	4232.2	466*	0.2	466*	0.6	466*	12537	0.5
rbg040a	42	539	386*	751.8	386*	738.1	386*	-	386*	16585	0.8
rbg041a	43	628	[382,417]	-	403	-	404	-	402*	29716	2.2
rbg042a	44	762	[409,435]	-	411*	149.8	411*	1036.4	411*	146107	35.6
rbg048a	50	1288	[455,527]	-	492	-	500	-	-	-	-
rbg049a	51	1083	[418,501]	-	488	-	492	-	-	-	-
rbg050a	52	1629	414*	18.6	414*	180.4	414*	95.6	-	-	-
rbg050b	52	1175	[453,542]	-	551	-	527	-	-	-	-
rbg050c	52	1396	[509,536]	-	543	-	544	-	-	-	-
rbg055a	57	765	814*	6.4	814*	2.5	814	-	814*	22509	2.4
rbg067a	69	843	1048*	5.9	1048*	4.0	1051	-	1048*	22200	3.2
rbg086a	88	927	[1049,1052]	-	1094	-	1086	-	1051*	15377	1.9
rbg092a	94	1367	[1102,1111]	-	1150	-	1109	-	1093*	44394	9.0
rbg125a	127	1824	1410*	229.8	1453	-	1446	-	1409*	32099	3.5
rbg132	132	1575	[1348,1400]	-	-	-	-	-	1360*	32080	6.1
rbg132.2	132	3126	[1069,1125]	-	-	-	-	-	-	-	-
rbg152	152	2125	[1770,1792]	-	-	-	-	-	1783*	41911	11.5
rbg152.3	152	6191	[1525,1594]	-	-	-	-	-	-	-	-
rbg172a	174	2837	[1787,1897]	-	-	-	-	-	1799*	234570	544.3
rbg193	193	3050	[2386,2452]	-	-	-	-	-	2414*	501396	3165.7
rbg193.2	193	6031	[1981,2093]	-	-	-	-	-	-	-	-
rbg201a	203	3287	[2159,2296]	-	-	-	-	-	2189*	244330	581.1
rbg233.2	233	7588	[2152,2304]	-	-	-	-	-	-	-	-
rbg233	233	3766	[2647,2786]	-	-	-	-	-	2689*	480384	1433.3

Similar to computational time for the instances by Pesant et al. [19], the DP algorithm solves some instances very quickly, for example 1.0 second for rbg034a. However, the DP spent more than two hours to obtain an optimal solution for rbg021.6, where the branch-and-cut algorithm [2] and hybrid constraint programming algorithm [13] only need a second. As mentioned before, it is difficult to compare the computational time due to the use of different processors, memory, computer languages, and operating systems. In addition, the algorithms perform differently on different instances.

Finally, note that the TSPTW problem can also be transformed into an elementary resource-constrained shortest path problem by including a large negative cost into each arc cost [1]. Then, the algorithms for

the elementary shortest path problem can be used to solve the TSPTW problem. However, due to the introduction of large negative costs, the triangular inequality property does not hold even if the triangular inequality holds in the original TSPTW problem. Hence, if the triangular inequality does not hold in the original TSPTW problem, one can just transform the TSPTW problem into the elementary shortest path problem and use the elementary shortest path algorithms [22; 23; 6] to solve it, which is as same as using DP directly designed for the TSPTW. When the triangular inequality holds in the original TSPTW problem, it is better to use the dynamic programming algorithm directly designed for the TSPTW in order to take the computational advantage from the triangular inequality.

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), with the focus on the computational study. 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. We also design a new label joining method so that labels can be joined at the same node in order to reduce computational time. If the triangular inequality holds, the dominance rules can be further strengthened and the corresponding computational time is substantially reduced.

The bi-directional resource-bounded label correcting algorithm performs well in solving TSPTW problems. In summary, (i) we obtain optimal solutions or show the infeasibility for all the instances by Dumas et al. [11] with the precision of four decimal places. We also solve certain difficult instances that were not optimally solved before; (ii) for the instances with relatively wide time windows by Gendreau et al. [14], we were able to generate optimal solutions for all the instances with 20 customers, most instances with 40 customers, and certain instances with 60 customers; (iii) 19 of the 31 Solomon instances by Potvin and Bengio [21] have been solved to optimality by the DP algorithm; (iv) we solved 18 Solomon instances by Pesant et al.[19] to optimality, five less than Focacci et al. [13]; and (v) for the asymmetric TSPTW instances by Ascheuer et al. [2], both the branch-and-cut algorithm [2] and hybrid constraint programming [13] solve 31 instances to optimality. The DP algorithm solves 36 instances to optimality, including certain instances with a large number of customers. To the best of our knowledge, no optimal solutions were reported for the instances [14; 21] and certain large instances [2]. We feel it is fair to say that our DP algorithm is competitive if compared with the state-of-art approaches in the literature.

We investigate the impacts of a large data precision on the instances by Gendreau et al. [14]. We find that the objective value with four decimal places is higher than the objective value without keeping decimal places, confirming the finding by Pesant et al. [19]. However, the computational time for the large precision

of four decimal places is similar to the one when the fractional travel times are simply truncated, even smaller in some cases.

Future research may investigate other TSP variations, such as the TSP with precedence constraints.

References

[1] Artigues C, Feillet D. A branch and bound method for the job-shop problem with sequence-dependent setup times. Annals of Operations Research 2008;159:135–159.

[2] Ascheuer N, Fischetti M, Grötschel M. Solving the asymmetric travelling salesman Problem with time windows by branch-and-cut. Mathematical Programming 2001;90:475–506.

[3] Baker EK. an exact algorithm for the time-constrained traveling salesman problem. Operations Research 1983;31(5):938–945.

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

[5] Beasley JE, Christofides N. An algorithm for the resource constrained shortest-path problem. Networks 1989;19(4):379–394.

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

[7] Calvo RW. A new heuristic for the traveling salesman problem with time windows. Transportation Science 2000;34(1):113–124.

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

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

[10] Dror M. Note on the complexity of the shortest-path models for column generation in VRPTW. Operations Research 1994;42(5):977–978.

[11] Dumas Y, Desrosiers J, Gelinas E, Solomon MM. An optimal algorithm for the traveling salesman problem with time windows. Operations Research 1995;43(2):367–371.

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

[13] Focacci F, Lodi A, Milano M. A hybrid exact algorithm for the TSPTW. INFORMS Journal on Computing 2002;14(4):403–417.

[14] Gendreau M, Hertz A, Laporte G, Stan M. A generalized insertion heuristic for the traveling salesman problem with time windows. Operations Research 1998;46(3):330–335.

[15] Kohl N, Desrosiers J, Madsen OBG, Solomon MM, Soumis F. 2-path cuts for the vehicle routing problem with time windows. Transportation Science 1999;33(1):101–116.

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

[17] Mingozzi A, Binaco L, Ricciardelli S. Dynamic programming strategies for the travelling salesman problem with time windows and precedence constraints. Operations Research 1997;45:365–377.

[18] Ohlmann JW, Thomas BW. A compressed-annealing heuristic for the traveling salesman problem with time windows. INFORMS Journal on Computing 2007;19(1):80–90.

[19] Pesant G, Gendreau M, Potvin JY, Rousseau JM. An exact constraint logic programming algorithm for the traveling salesman problem with time windows. Transportation Science 1998;32(1):12–29.

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

[21] Potvin JY, Bengio S. The vehicle routing problem with time windows part II: genetic search. INFORMS Journal on Computing 1996;8(2):165–172.

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

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

[24] Rochat Y, Taillard E. probabilistic diversification and intensification in local search for vehicle routing. Journal Of Heuristics 1995;1:147–167.

[25] Savelsbergh MWP. Local search in routing problems with time windows. Annals of Operations Research 1985;4(1):285–305.

[26] Solomon MM. Algorithms for the vehicle-routing and scheduling problems with time window constraints. Operations Research 1987;35(2):254–265.

[27] Taillard E, Badeau P, Gendreau M, Guertin F, Potvin J. A new neighborhood structure for the vehicle routing problem with time windows. Technical Report CRT-95-66 Centre de recherche sur les transports, Université de Montréal 1995.

[28] Thomas BW. Traveling Salesman Problem with Time Windows - Benchmark Data Sets. http://myweb.uiowa.edu/bthoa/TSPTWBenchmarkDataSets.htm 2008.

Appendix

Algorithm 1 Pseudo Code of the Bi-directional DP Algorithm

```
//Initialization
T ← max_{i∈N}{b_i + t_{i,n+1}}
T_b ← T/2
//store the initial forward label into linked list ForL and the initial backward label in linked list BackL
//ForL and BackL can be accessed by any function
ForL ← {(0, 0, 0)}
BackL ← {(0, T, n + 1)}
while true do
  //ForL is processed before BackL
  L ← ForL ∪ BackL
  ForL ← ∅
  BackL ← ∅
  if L = ∅ then
    //termination
    break
  end if
  //process forward labels before backward labels
  remove label l = (V, s, i) from the head of L
  //forward label extension
  if l is a forward label then
    for all out-going node j of i do
      if V_j = 0 then
        s' = max{s + t_ij, a_j}
        V_k' = V_k if k ≠ j, and V_j' = V_i + 1
        if s + t_ij ≤ T_b and s' ≤ b_j then
          DominanceChecking(V', s', j)
        end if
      end if
    end for
    //backward label extension
  else
    for all in-coming node j of i do
      if V_j = 0 then
        s' = min{s − t_ji, b_j}
        V_k' = V_k if k ≠ j, and V_j' = V_i + 1
        if s − t_ji ≥ T_b − G and s' ≥ a_i then
          DominanceChecking(V', s', j)
        end if
      end if
    end for
  end if
end while
//Label joining
for all node i from 0 to n + 1 do
  sort the forward labels of i according to |V|
  sort the backward labels of i according to |V|
  for all forward label l = (V^f, s^f, i) of i do
    use the binary search to determine the potential backward labels n = (V^b, s^b, i) so that |V^f| + |V^b| = n + 2
    for all potential backward label n = (V^b, s^b, i) do
      if all the nodes are covered only and only once by l and n and s^f ≤ s^b then
        obtain a TSP route with the cost c(V^f, s^f, i) + c(V^b, s^b, i).
      end if
    end for
  end for
end for
output the TSP route with the minimal cost
```

Algorithm 2 DominanceChecking(l=(V, s, i)): Label dominance sub-route

```
//use ⊆ instead of = for U if the triangular inequality holds
if l is a forward label then
  for all forward label n = (V̄, s̄, i) of i do
    if s ≤ s̄ and c(V, s, i) ≤ c(V̄, s̄, i) and U = Ū then
      label n is dominated by l and dropped from L
    end if
    if s̄ ≤ s and c(V̄, s̄, i) ≤ c(V, s, i) and Ū = U then
      label l is dominated by n
      //do not need to scan other labels
      return
    end if
  end for
else
  for all backward label n = (V̄, s̄, i) of i do
    if s ≥ s̄ and c(V, s, i) ≤ c(V̄, s̄, i) and U = Ū then
      label n is dominated by l and dropped from L
    end if
    if s̄ ≥ s and c(V̄, s̄, i) ≤ c(V, s, i) and Ū = U then
      label l is dominated by n
      //do not need to scan other labels
      return
    end if
  end for
end if
if l is a forward label then
  insert l into ForL
else
  insert l into BackL
end if
```