◈ Research Papers

An Algorithm for the Traveling Salesperson Problem with Time Windows and Lateness Costs

Suh, Byung Kyoo*
Kim, Jong Soo**

Abstract

This paper presents a model and dynamic programming based algorithm for the Traveling Salesperson Problem with Time Windows (TSPTW). The main difference of our model compared with the previous ones lies in that the time windows we are concerned are far more flexible and realistic. In the typical TSPTW, the service at a node must begin within the time grid called the time window that is defined by the earliest and the latest time to start the service at the node. But, in real business practices, a lateness cost is usually penalized rather than the service is prohibited at all when a vehicle arrives after the latest time.

Considering this situation, we propose a model with a new time window that allows an arrival after the latest time and penalizes the late arrival by charging a lateness cost. An algorithm introduced for the model is extensively tested to verify the accuracy and efficiency.

1. Introduction

1.1. Problem Statement and Research Objective

Traveling Salesperson Problem with Time Windows (TSPTW) is to find the minimum cost path that starts from and returns to a starting node, visiting each node within a time window once and only once. Time window here means the time grid that is defined by the earliest time, when a service may begin, and by the latest time, beyond which no visiting is allowed. Therefore, all of the previous research assumes that, if we arrive at a node earlier than the earliest time, we should wait until the earliest time. On the other hand, if we try to visit a node after the latest time, we are not permitted to visit the node at all.

However, in real business practice, when we try to visit a node after the latest time, we are usually allowed to start a service. But, at the same time, we have to pay for the late arrival. This lateness cost ranges from the profit loss on the sale to loss of goodwill of customer. This kind of business practice can be found almost in all industry including mail delivery, industrial waste collecting, school bus routing, A/S visiting business, etc. To reflect this real world situation more precisely, we are proposing a model with a new time window and solution methodology for the model.

* Ph. D. Course in Dept. of Industrial Engineering, Hanyang University
** Associate Professor in Dept. of Industrial Engineering, Hanyang University

1.2. Literature Review

Traveling Salesperson Problem(TSP) originated from the observation of movement of knight on chess plate by Euler and Vandermonde in 1759 and had become a research topic when Hamilton suggested Hamiltonian Cycle in 1856. Since that time, the problem has been investigated by many researchers due to its broad applicability and to its difficulty in obtaining an optimum solution.

Recently, several variants of the original TSP have been emerged as new research topics. These new variants include CTSP(constrained traveling salesperson problem, PCTSP(precedence-constrained traveling salesperson problem), and TSPTW [1]. (For more thorough review of TSP, the readers are referred to [11].) Among them, TSPTW, which is the research topic of this paper, is different from the original TSP in that visiting to each node should be made within a time grid called the time window [3, 4, 8, 9].

Since TSPTW is more realistic and is a background for more advanced models like vehicle routing problem, there have been a significant amount of researches. Savelsbergh[10] proved that finding a feasible solution to TSPTW is NP-complete and suggested an interchange heuristic to find an efficient feasible solution. Baker[2] suggested a branch and bound method and showed his algorithm could solve the problem with 50 nodes. Dumas et. al. [7] presented a dynamic programming based algorithm with reduced state, step, and state transition. Desrosiers et. al [5] presented a method to find an optimum solution for the dial-a-ride problem and showed that it solves a problem with 80 nodes within 6 seconds using CYBER 173.

All previous researches related with TSPTW do not accept the arrivals after the latest time, However, when we consider real business practice of industry, it is reasonable to allow the arrivals after the latest time and penalizing it by a lateness cost. Therefore, in this paper, we propose a new time window that allows and penalizes a visit after the latest time and introduce a dynamic programming based algorithm to find a good solution efficiently.

Thus our contribution to TSPTW field lies in that we are introducing a new time window model and an algorithm to find a solution for the model with realistic size. The rest of the article is organized as follows. In Section 2, we describe the model and introduce the algorithm for the model. In Section 3, we perform computational experiments to evaluate the accuracy and efficiency of the algorithm. Lastly, we summarize the results of the study and identify some areas for future research.

2. Model and Algorithm

2.1. Notation

We define the following notation.

$[a_j, b_j]$: time window at node j,

a_j : earliest time in the time window at jth node.

b_j : latest time in the time window at jth node.

t_j : time service actually started at jth node.

s_j : time required to complete service at node j,

l_j : lateness cost at jth node.

a_j : lateness cost per unit time at jth node,

U : delay time limit for all nodes.

c_{jk} : traveling cost from node j to node k.

τ_{jk} : traveling time from node j to node k.

2.2 Proposed time window

The structure of the proposed time window is shown in Figure 1. When we arrive at node j within its time window, there is no lateness cost. When we arrive at node j after the latest time, we have to pay for lateness costs of $a_j(t_j - b_j)$. Additionally, there exists a single valued maximum delay time limit, U, that applies to all nodes. This limit is to put restriction upon the maximum allowable delay to reflect the situation where there exists a practical limitation for delay. For example, a delivery to a grocery shop should be made no later than the shop's closing time.

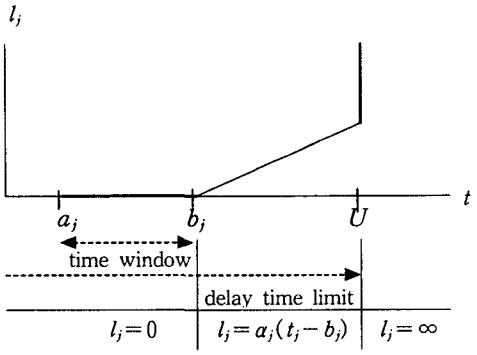

Figure 1. Proposed Time Window

Other important assumptions of the model are as followings:

(1) Cost matrix between nodes is $d_{ij} = d_{ji}$ for all nodes.

(2) Lateness cost occurs based on the visiting time rather than the service completion time. This assumption is for simplicity of presentation and can be relaxed without difficulty.

(3) The maximum allowable delay time limit does not apply to the returning time to a starting point.

2.3. The Outline of Algorithm

We are not able to use any of the existing algorithms for the proposed model due to the dissimilarity in time windows. Since the TSPTW is NP-complete and some application areas of it require a good solution in real time, we have developed a heuristic algorithm. The algorithm we are going to introduce is composed of two phases. In phase I of the algorithm, we are searching for the path with the smallest traveling cost. In this phase, we are considering neither the visiting time nor the lateness cost. In phase II, we find a candidate solution based upon the results of the phase I and lateness cost. This candidate is used to exclude all the inferior paths from further consideration as in branch and bound

method. Thus the basic mechanism of the phase II can be described as a conceptual application of the branch and bound method. Now, we explain the algorithm formally.

2.3.1. Phase I

Let

N = number of cities,

N_j = $\{2,3,\ldots,j-1,j+1,\ldots N\}$,

S = a subset of N_j ,

$f_i(j, S)$ = the traveling cost of the minimum path from node 1 to node j via the set of i intermediate nodes S.

Then equation (1) is the recurrence equation and equation (2) is boundary condition.

$$f_i(j, S) = \min_{k \in S} [f_{i-1}(k, S-\{k\}) + d_{kj}], \quad i=1,2,\ldots,N-2;\ j \neq 1;\ S \subseteq N_j . \tag{1}$$

$$f_0(j, \_) = d_{1j} \tag{2}$$

We use equations (1), (2) to calculate the value of $f_i(j, S)$ for each halfway node. Then we apply equations (3) or (4) to find the traveling cost for a complete path. For example, to find the minimum traveling cost of 1-2-3-4-5-6-1 path, we calculate the smallest traveling costs for 1-2-3-4 path and 1-6-5-4 path by (1) and (2) and, using (3), we find the cost of the original path.

$$\{f_{(N-2)/2}(j, S) + f_{(N-2)/2}(j, N_j - S)\} \text{ for } N = \text{even number} \tag{3}$$

$$\{f_{(N-3)/2}(j, S) + f_{(N-1)/2}(j, N_j - S)\} \text{ for } N = \text{odd number} \tag{4}$$

Note that equations (3) and (4) are based on the doubling-up method of dynamic programming. Our method is different from the ordinary doubling-up method in that we utilize the idea of designating a halfway node to find minimum cost of a path more efficiently. Computational procedures for two methods are different from each other. For more detailed discussion of the doubling-up procedure, please refer to [6].

It is known that the doubling-up procedure requires less than half of the calculation needed for ordinary formulation. Our algorithm is also able to enhance the computational efficiency partially due to the successful implementation of the basic idea of the doubling-up method.

2.3.2. Phase II

In Phase II, we find the path with the minimum total cost. (By the total cost, we mean the traveling plus lateness costs.) To do that, first, we rearrange the paths in increasing order of the total cost and call them $TC_1, TC_2, \ldots$, etc. Thus we denote the path with the least traveling cost as TC_1. Then we add the lateness cost to the traveling cost to get the total cost of TC_1. To make sure that the path is a feasible one, we check the visiting time to each node is before U. If it is true, we regard the total cost as a temporary solution. On the other hand, if the path is infeasible, then we consider the remaining path in the order until we find the first feasible path. The temporary solution found here is used as

an upper bound for the true optimum solution. Thus we are able to exclude the paths that have a traveling cost higher than the temporary solution. We continue this branch and bound like procedure until completion.

Here, we need to pay attention to the fact that the total cost of a path can be different from that of reversed visiting order. For example, path, $\eta_1-\eta_2-\eta_3-\cdots-\eta_n-\eta_1$ and its reversed path, $\eta_1-\eta_n-\eta_{n-1}-\cdots-\eta_2-\eta_1$ might have different total costs even though they have the same traveling cost. Thus, in Phase II, we ought to consider the reversed path before moving to the next lowest cost path. We present the algorithm based upon these ideas below.

2.4. Algorithm

Phase I:

Step 1. Find traveling cost for each halfway path using (1) and (2).
Step 2. Find complete path and its cost using (3) or (4).

Phase II:

Step 1.Rearrange the paths from phase I in increasing order of traveling cost. The paths arranged and their reversed paths will be referred to as a path group. Let the lowest cost path be the current path. Temporary solution $\zeta \leftarrow \infty$.

Step 2. (2.1) Calculate the total cost for the current path.
(2.2) If there exists at least one node to be visited later than U or if (the total cost of the current path $\geq \zeta$), then go to Step 3.
(2.3) Let $\zeta \leftarrow$ total cost of the current path.Remove the current and the paths with traveling cost higher than ζ and their corresponding reverse paths from the path group.

Step 3. (3.1) If there exists no more path in the path group, go to step 5.
(3.2) If the previous current path is in normal direction, let reverse direction of the path be the new current path and go to (2.1)
(3.3) Let the normal direction path with the next lowest cost be the new current path and go to (2.1).

Step 4. If $\zeta = \infty$, then the problem has no feasible solution. Otherwise, current temporary minimum cost and its path are final answers to the problem.

3. Numerical Example

Consider a TSP with the cost and traveling time matrices in Tables 1 and 2. In this example, we assume lateness cost coefficient $\alpha_j=2$, and lateness time limit $U=22$. Starting point is node 1 and time window and service time are given in Table 3.

Table 1. Cost Matirix

	1	2	3	4	5	6
1	-	13	15	8	3	20
2		-	18	24	9	12
3			-	12	21	5
4				-	16	11
5					-	7
6						-

Table 2. Traveling Time Matrix

	1	2	3	4	5	6
1	-	0.5	2.5	1	1	1.5
2		-	1.5	6	1.5	1.5
3			-	1	0.5	1.5
4				-	4	0.5
5					-	0.5
6						-

Table 3. Time Windows and Service Time

Node	Time Window $[a_j, b_j]$	Service Time s_j (minutes)
1	08~22	0
2	09~11	20
3	10~15	10
4	12~16	30
5	11~17	5
6	15~18	45

Table 4. Result after Step 1 of Phase II

sequence	equation	path	cost
①	$f_2(6,\{2,5\})+f_2(6,\{3,4\})$	1-5-2-6-3-4-1	49
②	$f_2(3,\{2,5\})+f_2(3,\{4,6\})$	1-5-2-3-6-4-1	54
③	$f_2(2,\{3,4\})+f_2(2,\{5,6\})$	1-4-3-2-6-5-1	60
④	$f_2(3,\{2,6\})+f_2(3,\{4,5\})$	1-2-6-3-4-5-1	61
⑤	$f_2(3,\{2,4\})+f_2(3,\{5,6\})$	1-2-4-3-6-5-1	64
⑥	$f_2(4,\{2,3\})+f_2(4,\{5,6\})$	1-2-3-4-6-5-1	64
⑦	$f_2(2,\{3,6\})+f_2(2,\{4,5\})$	1-3-6-2-5-4-1	65
⑧	$f_2(6,\{2,3\})+f_2(6,\{4,5\})$	1-2-3-6-4-5-1	66
⑨	$f_2(4,\{2,5\})+f_2(4,\{3,6\})$	1-5-2-4-6-3-1	67
⑩	$f_2(5,\{2,3\})+f_2(5,\{4,6\})$	1-3-2-5-6-4-1	68
⑪	$f_2(5,\{2,4\})+f_2(5,\{3,6\})$	1-4-2-5-6-3-1	68
⑫	$f_2(4,\{2,6\})+f_2(4,\{3,5\})$	1-2-6-4-3-5-1	72
⑬	$f_2(2,\{3,5\})+f_2(2,\{4,6\})$	1-5-3-2-6-4-1	73
⑭	$f_2(5,\{2,6\})+f_2(5,\{3,4\})$	1-2-6-5-3-4-1	73
⑮	$f_2(6,\{2,4\})+f_2(6,\{3,5\})$	1-4-2-6-3-5-1	73

After step 1 of phase II, we obtain the results summarized in Table 4. We illustrate the calculation for path 1-5-2-6-3-4-1 below.

$$f_2(6,\{2,5\}) = \min[f_1(2,\{5\}) + d_{26},\ f_1(5,\{2\}) + d_{56}] = \min[f_0(5,-) + d_{52} + d_{26},\ f_0(2,-) + d_{25} + d_{56}]$$
$$= \min[d_{15} + d_{52} + d_{26},\ d_{12} + d_{25} + d_{56}] = \min[3+9+12,\ 13+9+7] \quad (5)$$
$$= \min[24,\ 29] = 24, \quad \text{PATH} = 1-5-2-6$$

$$f_2(6,\{3,4\}) = \min[f_1(3,\{4\}) + d_{36},\ f_1(4,\{3\}) + d_{46}] = \min[f_0(4,-) + d_{43} + d_{36},\ f_0(3,-) + d_{34} + d_{46}]$$
$$= \min[d_{14} + d_{43} + d_{36},\ d_{13} + d_{34} + d_{46}] = \min[8+12+5,\ 15+12+11] \quad (6)$$
$$= \min[25,\ 38] = 25, \quad \text{PATH} = 1-4-3-6$$

Note that (5) and (6) are derived using (1) and (2). Now, by (3), the minimum traveling cost of 1-5-2-6-3-4-1 is given as $f_2(6, 2,5) + f_2(6, 3,4) = 24 + 25 = 49$.

Phase II starts with path ① as shown in Table 5. The total cost of the path turns out to be 65.50. ζ is set to this value. We find that paths ⑧~⑮ have traveling cost larger than ζ and remove them from the path group. Now, we calculate the total cost of ①'s reverse path. It is also larger than ζ and the path is removed. Next, we find that ② has 62.67. It replaces the current ζ. Additionally, we can remove paths ⑤~⑦ from the list. Since the remaining paths either have larger total cost or are infeasible, we stop to find path ② is solution of the algorithm. The solution 1-5-2-3-6-4-1 with total cost 62.67 turns out to be optimal for the problem.

Table 5. Result of the Algorithm

processing order	path	total cost	ζ
1	①	65.50	65.50
2	①'s rev. dir.	71.167	65.50
3	②	62.67	62.67
4	②'s rev. dir.	85.00	62.67
5	③	80.17	62.67
6	③'s rev. dir.	95.17	62.67
7	④	∞	62.67
8	④'s rev. dir.	92.17	62.67

4. Computational Experiment

We perform experiment on a Pentium II 333 PC system using a program coded in Object Pascal. The primary objective of the experiment is to test the accuracy of the algorithm. The secondary one is to observe how fast it is. The third objective is to know if there exists a relationship between the width of time windows and the accuracy of the algorithm. To do this, we compare the solutions and computation times of the algorithm and a complete enumeration. The complete enumeration is used here because there exists no previous algorithm suitable for our problem. We make 20 runs for each problem size to obtain an averaged performance.

The probability distributions to generate necessary parameters are as follows:

Traveling time between two nodes ~ $(12/N)*U$ [0.4, 0.6],
Traveling cost between two nodes ~ traveling time $*Uniform$[0.8, 1.2],
Service time at a node ~ $(12/N)*U$ [0.2, 0.4],
Lateness cost ~ average traveling cost $*$ $Uniform$[0.1, 10],
Earliest starting time at a time window ~ $Uniform$ [8, 20],
Latest starting time at a window ~ earliest starting time at the same window + $Uniform$ [1, 5],

Maximum delay time limit = 20.

In accuracy test shown in Table 6 and Figure 2, it is observed that the percent deviation from the optimal is no more than 1.53 percent on average.

Table 6. Percent Deviation of the Algorithm (unit : %)

number of node	average percent deviation	max. percent deviation	number of optimum found
5	0.00	0.00	20
6	0.00	0.00	20
7	0.00	0.00	20
8	0.11	0.67	18
9	0.00	3.02	19
10	1.53	3.24	18

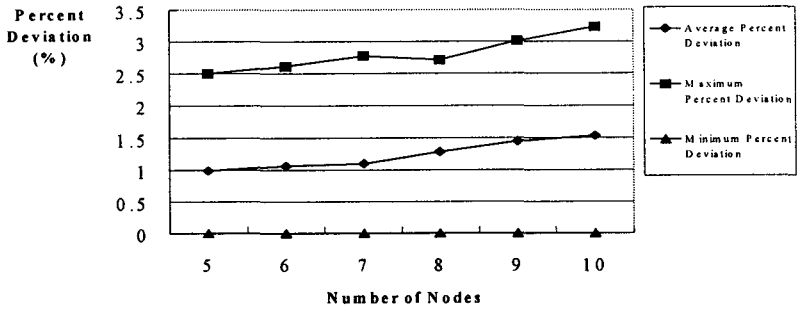

Figure 2. Computational Behavior of the Algorithm for Various Sizes of Nodes

Table 7 and Figure 3 show the computation times for the two methods. As we have expected, the time for the complete enumeration grows exponentially and seems to reach a practical limit at 10 nodes problem. Meanwhile, the proposed algorithm needs more physical storage spaces as the problem size increases.

Lastly, we did experiment to check if there exists any relationship between the width of time windows and the accuracy of the algorithm. For this purpose, we measure the percent deviation of the solution as we increase the width of the window from 2 to 5 hours. We averaged result of problems ranging from 5 to 10 nodes. The results are summarized in Table 8 and Figure 4.

Table 7. Computation Times for Various Sizes of problems (unit : CPU second)

numbers of node	complete enumeration	proposed algorithm
5	1.1	0.5
6	1.5	0.8
7	15.3	3.1
8	111.8	8.8
9	1,218.8	27.3
10	22,390.2	181.1

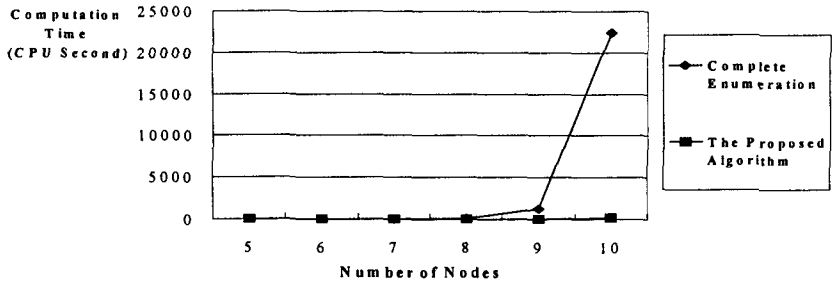

Figure 3. Comparison of Computation Times

Table 8 shows that, as the width of time window grows wider, the percent deviation decreases. Since the proposed algorithm guarantee to find the optimum solution for the problem without lateness cost, it generates more accurate solution as the effect of lateness cost diminishes.

Table 8. Average Percent Deviation of the Algorithm for Various Sizes of the Time Windows (unit : %)

ave. time length of time window	percent deviation	imp. rate*
2 hrs	2.11^{E_1}	-
3 hrs	1.05^{E_2}	50.24
4 hrs	0.39^{E_3}	81.52
5 hrs	0.24^{E_4}	88.63

* improvement rate = $[(E_1 - E_i) / E_1] \times 100$, where $i = 2, 3, 4$.

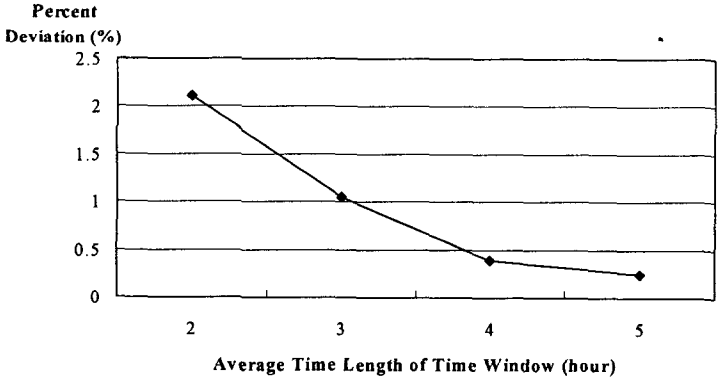

Figure 4. Computational Behavior of the Algorithm for Various Intervals of the Time Windows

In summary, the proposed algorithm is capable of finding accurate solution in reasonable amount of time. We also found that the maximum solvable problem size is limited only by the memory size of a computer. Also noted is that its accuracy is improved as the time windows become wider. This is contrary to the existing algorithms. Because the problem

with a wider time window is considered more general and realistic [8], we may conclude that the algorithm possess a very desirable property. What should be also mentioned is the algorithm's flexibility. The algorithm can be easily modified to handle the several kinds of variants of TSPTW including the problems with the early arrival penalty and nonlinear lateness cost.

5. Conclusion

In this paper, we propose a TSPTW model with a new time window that allows and penalizes visit after latest time. A dynamic programming based algorithm that we have introduced proves to find good solution in resonable time. Thus, we believe that the method should be a good addition to the research in TSPTW and hope that it would be used to tackle real world problems.

As future research, an integer programming model and an algorithm based on it could be developed. Applications to more complex topics such as the vehicle routing problem is another example of potential extentions.

References

1. Bodin, L. D. & B. L. Golden, "Classification in Vehicle Routing and Scheduling," Networks, Vol. 11, pp. 97-108, 1981.
2. Baker, E. K, "An Exact Algorithm for the Time-constrained Traveling Salesman Problem," Opns. Res., Vol. 31, pp. 938-945, 1983.
3. Solomon, M. M., "Algorithms for the Vehicle and Scheduling Problems with Time Window Constraints," Opns. Res., Vol. 5, pp. 254-265, 1987.
4. Desrochers, M., J. Desrosiers, and M. Solomon, "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows," Opns. Res., Vol. 40, pp. 342-354, 1992.
5. Desrosiers, J., Y. Dumas, and F. Soumis, "A Dynamic Programming Solution of the Large-Scale Single-vehicle Dial-a-ride Problem with Time Windows," American J. Math. and Mgmt. Sci., Vol. 6, pp. 301- 325, 1986.
6. Dreyfus, S. and A. Law, The Art and Theory of Dynamic Programming, Academic Press, Inc., 1977.
7. Dumas, Y., J. Desrosiers, E. Gelinas, and M. Solomon, "An Optimal Algorithm for the Traveling Salesman Problem with Time Windows," Opns. Res., Vol. 43, pp. 367- 371, 1995.
8. Gendreau, M., A. Hertz, G. Laporte, and M. Stan, "A Generalized Insertion Heuristic for the Traveling Salesman with Time Windows," Opns. Res., Vol. 43, pp. 330- 335, 1998
9. Mingozzi, A., L. Bianco, S. Ricciadelli, "Dynamic Programming Strategies for the Traveling Salesman Problem with Time Window and Precedence Constraints," Opns. Res., Vol. 45, pp. 365-377, 1997.
10. Savelsbergh, M., "Local Search in Routing Problems with Time Windows," Anns. Opns. Res., Vol. 4, pp. 285-305, 1985.
11. Mark, H. N., "The Traveling Salesman Problem - a Review of Theory and Current Research," http://www.ececs.uc.edu/~mnoschan/sale.html.