Journal of the Operations Research
Society of Japan
Vol. 39, No. 1, March 1996

A SHORTEST PATH PROBLEM ON THE NETWORK WITH AGV-TYPE TIME-WINDOWS

Ryusuke Hohzaki *National Defense Academy*
Susumu Fujii *Kobe University*
Hiroaki Sandoh *University of Marketing & Management Science*

(Received February 28, 1994; February 16, 1995)

Abstract This paper deals with the shortest path problem on a network with time-windows. The concept of time-windows is recently introduced by Desrosiers et al. as a subproblem of the traveling salesman problem involving time constraints. Authors consider another type of time-windows, which we name the AGV-type time-window, and define the shortest path problem on the network with the AGV-type time-windows. We verify that this problem has a structure similar to the conventional shortest path problem without time-windows and propose several methods to solve the shortest path problem with such time-windows. The method consists of two procedures: first, we divide the main problem into subproblems, and then solve these subproblems using the dynamic programming or the branch and bound method. Numerical experiments are carried out to compare the computational performance of the proposed algorithms.

1. Introduction

We can deal with a routing problem of automated guided vehicles (AGVs) in production system as a shortest path problem on a transfer network. A collision-free path of an AGV must be determined considering the whole traffic of other AGVs[6]. These traffic data can be expressed as the time constraints imposed on nodes and arcs of the network. When the AGV is passing through a node or an arc, the other AGVs cannot gain access to the same place. On the other hand, during time periods when no AGV is occupying a place, every AGV is free to pass through there. When we call time periods the time-window, the routing problem of the AGV can be defined as the shortest path problem with time-windows (SPPTW)[7].

Desrosiers[2] and Desrochers[4] investigated the SPPTW as a subproblem in solving the traveling salesman problem with time constraints and Solomon[8] surveyed it from the view point of the vehicle routing. Characteristics of their time-windows were: (1) time-windows exist only on nodes, and (2) a time-window on node, say node i, has only one time interval, say $[c_i, d_i]$, and c_i does not give any constraint on vehicle's arrival time. However, considering the traffic system of AGVs on the transfer network, it is clear that the time-window for such a system must have the following characteristics: (1') time-windows may exist either on arcs or nodes, (2') time-windows generally consist of several time intervals, and (3') an AGV arriving at a node or an arc during a certain time interval of the time-window must leave there during the same time interval. In order to distinguish two types of time-windows, we term the latter type of time-windows the AGV-type time-windows. For the SPPTW with the former type of time-windows, Desrosiers et al.[3] developed an algorithm using the dynamic programming. Desrochers and Soumis[4] proposed a more efficient method. However, when we discuss the collision-free shortest time route of the AGV on the transfer network, it is inevitable to consider not Desrosiers' time-windows but AGV-type time-windows. We[7] first introduced the AGV-type time-windows to the routing control problem of AGVs in factories, which was problem-oriented. In this paper, we purified the AGV-type time-windows concept and theoretically investigate the SPPTW on the network with AGV-type time-windows, where arcs are not necessarily of positive length. Furthermore, we propose three algorithms to solve the SPPTW, Bellman-Ford-like method, Yen-like method and Branch-and-Bound-like method.

One of other examples with AGV-type time-windows is the traffic problem to find the shortest time route on the road, where the road has some places under construction and the time schedule of the construction is given as time-windows. Another field which the concept of the AGV-type time-windows could be applied to is the project scheduling problem. Traditional PERT method was developed on an arrow diagram representing the detail of the project. If job elements on the diagram have pre-occupied schedules in advance, a decision maker must make a plan which finishes the project as early as possible without intervening the original schedule. In this case, the project scheduling problem is considered as the SPPTW on the diagram with AGV-type time-windows.

In the next section, we modify an original network and formulate SPPTW on the modified network. In Section 3, we propose three algorithms to solve the SPPTW, with their computational complexities being evaluated. Numerical examples are given in Section 4 to compare the computational performances of three algorithms proposed in this paper.

2. Shortest Path Problem with AGV-Type Time-Windows

2.1 AGV-type shortest path problem

Let $G(N, A)$ be a network with the set of nodes N and the set of directed arcs A. An arc (i, j) is directed from node i to node j, with length d_{ij} which is possibly negative. Now, an AGV leaves a start node s after time t_0 and arrives at a destination node e, satisfying the time-window constraints on its way. The time-windows given on node i and the entrance of arc (i, j) are denoted as $T(i)$ and $S(i, j)$, respectively. $T(i)$ and $S(i, j)$ are represented as a set of line segments as follows.

(2.1) $T(i) = \{T_k^i = [t_{k1}^i, t_{k2}^i];\ k = 1, \cdots, n_i\}$,

(2.2) $S(i,j) = \{S_k^{ij} = [s_{k1}^{ij}, s_{k2}^{ij}];\ k = 1, \cdots, m_{ij}\}$,

where

(2.3) $t_{k1}^i \le t_{k2}^i,\ k = 1, \cdots, n_i,\ \ t_{k-12}^i < t_{k1}^i,\ k = 2, \cdots, n_i$,

(2.4) $s_{k1}^{ij} \le s_{k2}^{ij},\ k = 1, \cdots, m_{ij},\ \ s_{k-12}^{ij} < s_{k1}^{ij},\ k = 2, \cdots, m_{ij}$.

The intervals T_k^i and S_k^{ij} are referred to as open time periods. If the AGV arrives at node i at time t, leaves there at time $t'(\ge t)$ and goes through arc (i, j), t and t' must satisfy $t_{k1}^i \le t \le t' \le t_{k2}^i$ and $s_{k'1}^{ij} \le t' \le s_{k'2}^{ij}$ for some k and k'. The start time t_0 is assumed to belong to an open time period of node s. The AGV-type SPPTW is the problem of finding the optimal route that gives the earliest arrival time of an AGV to the destination node e, that is, finding the minimum arrival time u_l at node e achieved by the sequential lines of nodes $\{r_0 = s, r_1, \cdots, r_l = e\}$ and the time schedule $\{u_k, v_k\}$ attached to node r_k satisfying $u_0 = t_0$, $u_l \in T(e)$ and $[u_k, v_k] \subseteq T(r_k)$, $v_k \in S(r_k, r_{k+1})$, $u_{k+1} = v_k + d_{r_k\ r_{k+1}}$ for $k = 1, \cdots, l-1$. We assume that the network has no cycle of negative length for the SPPTW to be non-trivial.

2.2 Definitions of operators on time-windows

Let $\boldsymbol{K}$ be the family of the sets of line segments defined by equations (2.1) and (2.2), and $\boldsymbol{R}$ denotes the set of real numbers. For an $r \in \boldsymbol{R}$, empty set $\emptyset$ and two elements $X = \{[x_1^k, x_2^k];\ k = 1, \cdots, p\}$ and $U = \{[u_1^h, u_2^h];\ h = 1, \cdots, q\}$ of $\boldsymbol{K}$, operators $MIN, \oplus, \ominus$, and $\otimes$ are defined as follows.

(2.5) $MIN \;\; X = x_1^1 \in \boldsymbol{R}$

(2.6) $MIN \;\; \emptyset = \infty$

(2.7) $X \oplus r = \{[x_1^k + r, x_2^k + r];\; k = 1, \cdots, p\} \in \boldsymbol{K}$

(2.8) $X \ominus r = \{[x_1^k - r, x_2^k - r];\; k = 1, \cdots, p\} \in \boldsymbol{K}$

(2.9) $X \otimes U = \{[x_1^k, x_2^k] \cap [u_1^h, u_2^h];\; k = 1, \cdots, p,\; h = 1, \cdots, q\} \in \boldsymbol{K}$

Relation $\ll$ is defined by the following equivalence relation.

(2.10) $U \ll X \Longleftrightarrow u_2^q < x_1^1$

For $r \in \boldsymbol{R}$ and $X \in \boldsymbol{K}$, $r \in X$ means that $r \in [x_1^k, x_2^k]$ for a certain $1 \leq k \leq p$.

2.3 Modification of the original network

Let us modify the structure of original network $G(N, A)$ by the following procedure.

(1) For node $i \in N$, create n_i copies of this node, where n_i is the number of open time periods on node i as shown in equation (2.1). The resulting nodes are denoted as i_k, $k = 1, \cdots, n_i$, and we associate the time-window T_k^i of equation (2.1) with node i_k.

(2) For directed arc $(i, j) \in A$, if for $1 \leq k \leq n_i$ and $1 \leq l \leq n_j$

(2.11) $W = T_k^i \otimes S(i,j) \otimes \left(T_l^j \ominus d_{ij}\right) \neq \emptyset$,

we add a new directed arc from node i_k to j_l, and assign the time-window W to the entrance of the arc.

The newly generated network by the above procedure is referred to as the modified network. Now we prove the equivalence of the modified network to the original one with respect to SPPTW. Departure of an AGV from node i during open time period T_k^i and arrival of the AGV at node j during T_l^j on the original network correspond to the movement from node i_k to node j_l on the modified network. The movement is subject to the time-windows $T(i)$ and $S(i,j)$ imposed on node i and an entrance of arc (i,j), respectively. Furthermore, in order to arrive at node j during $T(j)$, the AGV must depart from node i during time periods $T(j) \ominus d_{ij}$. The relation (2.11) implies that the AGV can travel from node i_k to node j_l on the modified network. However, it is impossible for the AGV to move from node i_k to node $i_l (k \neq l)$, which is stated as the characteristic (3') of the AGV-type time-window in Section 1. Thus, there exists no arc between node i_k and i_l $(k \neq l)$ on the modified network. Only an open time period belongs to each node.

The total number of nodes on the modified network is identical to that of open time periods attached to the nodes of the original network. With respect to the total number of arcs, we obtain the next theorem.

Theorem 1. *The total number of arcs between node i_k and j_l $(k = 1, \cdots, n_i,\; l = 1, \cdots, n_j)$ on the modified network is at most $n_i + n_j - 1$. Therefore, the maximum number of arcs on the modified network is given by*

(2.12) $\displaystyle\sum_{(i,j)\in A} (n_i + n_j - 1)\,.$

Proof: See Appendix. □

In the procedure of generating the modified network from the original one, (2.11) is repeatedly evaluated for all arcs (i,j) of the original network. Binary operation $\otimes$ or $\ominus$ include the arithmetic of comparison or subtraction, respectively. Consequently, the computational complexity of generating the modified network is O($\sum_{(i,j)\in A}(n_i + m_{ij} + 2n_j)$).

2.4 Transformation of AGV-type SPPTW into subproblems

From the arguments in the previous section, the original AGV-type SPPTW is able to be transformed into the following problem on the modified network.

Assume that $t_0 \in T_p^s$, $1 \leq p \leq n_s$. The original SPPTW is equivalent to the problem of finding the route which allows the AGV to depart from node s_p after time t_0 and arrive as early as possible at either one of the nodes e_k for $1 \leq k \leq n_e$, on the modified network. This problem is decomposed into n_e subproblems with destination nodes e_k, $k = 1, \cdots, n_e$. Each subproblem is the SPPTW with one open time period on each node. Since time periods $T_1^e, \cdots, T_{n_e}^e$ on destination node e of the original network satisfy $T_1^e \ll T_2^e \ll \cdots \ll T_{n_e}^e$, all n_e subproblems need not to be solved. Namely, we solve the subproblem with destination node e_k on the modified network sequentially for $k = 1, \cdots, n_e$ and the solution we find first gives the optimal route to the original problem.

From now on, we discuss how we solve the subproblem with time-window constraints of one open time period at each node. For the modified network, we use the following notations without loss of generality, which appeared in the original network also. We denote the total number of nodes of the modified network by n, and number all nodes from 1 to n, of which node 1 indicates the start node and n the destination node. And let the set of nodes, the set of arcs, the time-window on node i and the time-window on arc (i, j) be denoted as N, A, $T(i)$ and $S(i, j)$, respectively. Furthermore, we specify the time period of $T(i)$ by an open time period

(2.13) $T(i) = [a_i, b_i]$.

3. Algorithms to Solve the Subproblems

3.1 Loops and subroutes of an optimal route

For the subproblem on the modified network , discussed in the previous section, the following theorems can be obtained.

Theorem 2. *If there exists an optimal route, there also exists an optimal route without loop.*

Proof: Let the sequence of nodes of an optimal route with loop at node j be $\{l_1 = 1, l_2, \cdots, l_k = j, \cdots, l_{k'} = j, \cdots, l_t = n\}$, and let the arrival time and departure times at node l_h of the route be, respectively, (u_h, v_h). From (2.13) and the non-negativeness of the length of loops, we obtain

(3.1) $a_j \leq u_k \leq v_k \leq u_{k'} \leq v_{k'} \leq b_j$.

Then, the new route $\{l_1 = 1, l_2, \cdots, l_k = j, l_{k'+1}, \cdots, l_t = n\}$ is optimal too, with the arrival and departure times being $(u_1 = t_0, v_1), (u_2, v_2), \cdots, (u_k, v_{k'}), (u_{k'+1}, v_{k'+1}), \cdots, (u_t, *)$. Thus, an optimal routes with no loops can be obtained from the one with loops. □

Theorem 3. *For an arbitrary optimal route, any of the subroute from start node* 1 *is optimal.*

Proof: As in Proof of Theorem 2, let $\{l_1 = 1, l_2, \cdots, l_t = n\}$ be an optimal route with the arrival and departure times at node l_h being (u_h, v_h). Assume that an optimal route from node 1 to an arbitrary node l_k of the route is $\{l'_1 = 1, l'_2, \cdots, l'_p = l_k\}$ with the arrival and departure times at node l'_m being (u'_m, v'_m). Since $u'_p \leq u_k$, a route $\{l'_1 = 1, l'_2, \cdots, l'_p, l_{k+1}, \cdots, l_t = n\}$ with the arrival and departure times $(u'_1 = t_0, v'_1), (u'_2, v'_2), \cdots,$
$(u'_p, v_k), (u_{k+1}, v_{k+1}), \cdots, (u_t, *)$ is optimal either. □

By Theorems 2 and 3, it suffices to search for an optimal route only among the set of routes without loops and with all subroutes being optimal. The above theorems suggest a method to solve subproblems. Namely, from Theorem 2, we know that an optimal route is made up of at most n nodes. Theorem 3 points out that an optimal route to a certain node can be found by searching for optimal routes to its adjacent nodes.

3.2 Algorithms

By theorems in the previous section, we present algorithms to find a shortest path such that an AGV starts from node 1 after time t_0, runs through it and arrives, as early as possible, at node n on the modified network. Concerning the shortest path problem on a traditional network without any time-windows, there are some famous algorithms such as Bellman-Ford method and Yen method. We show that the similar algorithms could be proposed for the AGV-type SPPTW.

(1) **Bellman-Ford-like method**

We can find a shortest path to node i by using the shortest paths to node k where $(k,i) \in A$. Assume that the earliest arrival time and the departure time at node k, $(k,i) \in A$ are u_k and v_k, respectively. The value v_k is at least as large as u_k and located within the time-windows $T(k)$ and $S(k,i)$. Departure from node k at time v_k implies arrival at node i at time $v_k + d_{ki}$. Thus, the earliest arrival time u_i at node i is determined by the following equation.

$$\begin{aligned}
(3.2) \quad u_i &= \min_{(k,i)\in A} \{\min\{v_k + d_{ki} \in T(i);\ v_k \in S(k,i),\ u_k \le v_k \le b_k\}\} \\
&= \min_{(k,i)\in A} \{\min\{v_k + d_{ki};\ a_i \le v_k + d_{ki} \le b_i,\ v_k \in S(k,i),\ u_k \le v_k \le b_k\}\} \\
&= \min_{(k,i)\in A} \{\min\{v_k + d_{ki};\ v_k \in T(i) \ominus d_{ki},\ v_k \in S(k,i),\ v_k \in [u_k, b_k]\}\} \\
&= \min_{(k,i)\in A} \{\min\{v_k + d_{ki}; v_k \in (T(i) \ominus d_{ki}) \otimes S(k,i) \otimes [u_k, b_k]\}\} \\
&= \min_{(k,i)\in A} \{MIN\ B_{ki} + d_{ki}\}
\end{aligned}$$

where B_{ki} is defined by

$$(3.3) \quad S'(k,i) = (T(i) \ominus d_{ki}) \otimes S(k,i)$$

$$(3.4) \quad B_{ki} = S'(k,i) \otimes [u_k, b_k].$$

Equation (3.2) gives the earliest arrival time u_i as well as a shortest path which is obtained by adding node i to the shortest path to node k with $u_i = MIN\ B_{ki} + d_{ki}$.

Now, we are ready to state an algorithm based on a recursive relation.

(i) Initialize variables as follows.

$$(3.5) \quad m = 0,\ u_1^{(0)} = t_0,\ u_i^{(0)} = \infty\ (i \ne 1)$$

(ii) For all $i \in N$,

$$(3.6) \quad u_i^{(m+1)} = \min\left\{u_i^{(m)}, \min_{(k,i)\in A}\left(MIN\ \ B_{ki}^{(m)} + d_{ki}\right)\right\}$$

where

$$(3.7) \quad B_{ki}^{(m)} = S'(k,i) \otimes \left[u_k^{(m)}, b_k\right]$$

(iii) Let $m = m + 1$ and go back to (ii).

Variable $u_i^{(m)}$ denotes the earliest arrival time at node i through directed paths with at most $m+1$ nodes. The above recursive algorithm terminates if none of the variables $u_i^{(m)}$, $i \in N$ change any longer. At that time, not only the earliest arrival time at the destination node, but also those at all nodes have been found. By Theorem 2, the algorithm stops in at most $m = n - 1$ steps. Comparison and addition are all arithmetic needed in (3.6). Therefore, the computational complexity of this algorithm is $O(2m_0n^3)$, where m_0 is the maximum number of time intervals on each arc.

A special case of

(3.8) $T(i) = [-\infty, +\infty], \quad i \in N,$

(3.9) $S(i,j) = [-\infty, +\infty], \quad (i,j) \in A$

gives a situation in which all nodes and arcs are always open to AGVs, that is there is no time-windows constraints. In this case, the SPPTW is reduced to the classical shortest path problem. Then, (3.6) becomes

(3.10) $u_i^{(m+1)} = \min \left\{ u_i^{(m)}, \min_{(k,i) \in A} \left(u_k^{(m)} + d_{ki} \right) \right\}$

which is nothing but the Bellman-Ford method[1][5]. For this reason, we name the proposed algorithm (3.5)~(3.7) the Bellman-Ford-like method or the B-F-like method for short.

(2) **Yen-like method**

For the classical shortest path problem without time-window constraints, Yen[9] improved the Bellman-Ford method to construct an algorithm with smaller memory and computational time requirements. His improvement can be applied directly to the above B-F-like method. Thus, we derive the following algorithm.

(i) Initialize variables as follows.

(3.11) $m = 0, \quad u_1^{(0)} = t_0, \quad u_i^{(0)} = \infty \ (i \neq 1)$

(ii) If m is even, calculate the following for $i = 1, \cdots, n$.

(3.12) $u_i^{(m+1)} = \min \left\{ u_i^{(m)}, \min_{\substack{k<i \\ (k,i) \in A}} \left(MIN \ \ B_{ki}^{(m)} + d_{ki} \right) \right\}.$

Otherwise, calculate the following for $i = n, \cdots, 1$.

(3.13) $u_i^{(m+1)} = \min \left\{ u_i^{(m)}, \min_{\substack{k>i \\ (k,i) \in A}} \left(MIN \ \ B_{ki}^{(m)} + d_{ki} \right) \right\}$

(iii) Let $m = m + 1$ and go back to (ii).

We name the above algorithm the Yen-like method because it becomes exactly the Yen method if we substitute $u_k^{(m)}$ for $MIN \ B_{ki}^{(m)}$ of equations (3.12) and (3.13). The condition for termination of the Yen-like method is the same as that for the B-F-like method, and its computational complexity is evaluated as $O(m_0 n^3/2)$.

(3) **Branch and bound-like method**

In the B-F-like or the Yen-like methods, the earliest arrival time at each node is revised recursively by the fundamental relation (3.2). The process converges in finite steps. Taking an alternative path, we can devise another algorithm.

In two algorithms proposed in the previous section, the revision of the earliest arrival time at node i is carried out by using $u_k^{(m)}$ on its backward adjacent nodes k, $(k,i) \in A$. In the following algorithm, a certain node i^* is selected first and for a forward adjacent node k, $(i^*, k) \in A$, the earliest time on that node is revised by $u_{i^*}^{(m)}$. Namely, for node i^*, the following updating is carried out.

(3.14) $u_k^{(m+1)} = \min \left\{ u_k^{(m)}, MIN \ \ B_{i^*k}^{(m)} + d_{i^*k} \right\}.$

If $u_k^{(m)}$ is revised by the second term of the right-hand side of (3.14), we call it as the transportation of the earliest arrival time at node i^* to node k. Such sequential transportation from the initial state of (3.11) produce an optimal route, which is the same as the above two methods.

If a selected $u_{i^*}^{(m)}$ is known to stay unchanged afterward, meaning that $u_{i^*}^{(m)}$ is the true earliest arrival time at node i^*, it is enough to calculate (3.14) only once. When the destination node is selected as such a node, the shortest path problem has been solved. The so-called A^* algorithm is available as a method of selecting such a node i^*.

Consider a real valued function $f : N \times \boldsymbol{R} \longrightarrow \boldsymbol{R}$ with the following properties:

(3.15) $f(i,t) \leq f(j, t + d_{ij}), \quad for \ (i,j) \in A,$

(3.16) $f(i,t) < f(i,t'), \quad for \ t < t'.$

The following could be used to select i^*.

(3.17) $f(i^*, u_{i^*}^{(m)}) = \min_{j \in N} f(j, u_j^{(m)})$

The value $u_{i^*}^{(m)}$ of a selected node i^* is never revised by (3.14). That can be proved by the reductive absurdity as follows. Assume that $u_{i^*}^{(m)}$ is revised by sequential transportations of $u_j^{(m)}$. Let the sequence of nodes, through which this transportation is made, be $L = \{j, l_1, l_2, \cdots, l_p, i^*\}$. The transportation is carried out by the second term of equation (3.14). Since

(3.18) $MIN \ B_{kl}^{(m)} + d_{kl} \geq u_k^{(m)} + d_{kl},$

for arbitrary nodes k and l, the transportation along L gives the following inequalities.

$$\begin{aligned} u_{l_1}^{(m+1)} &= MIN \ B_{jl_1}^{(m)} + d_{jl_1} \geq u_j^{(m)} + d_{jl_1}, \\ u_{l_2}^{(m+2)} &= MIN \ B_{l_1 l_2}^{(m+1)} + d_{l_1 l_2} \geq u_{l_1}^{(m+1)} + d_{l_1 l_2} \geq u_j^{(m)} + d_{jl_1} + d_{l_1 l_2}, \\ &\vdots \end{aligned}$$

Thus, by the whole transportation, a certain earliest arrival time larger than $u_j^{(m)} + d_{jl_1} + d_{l_1 l_2} + \cdots + d_{l_p i^*}$ is realized. Since the earliest arrival time revises $u_{i^*}^{(m)}$,

(3.19) $u_{i^*}^{(m)} > u_j^{(m)} + d_{jl_1} + d_{l_1 l_2} + \cdots + d_{l_p i^*}.$

Furthermore, from (3.15) and (3.16), the following is derived.

$$\begin{aligned} (3.20) \ f(i^*, u_{i^*}^{(m)}) &> f(i^*, u_j^{(m)} + d_{jl_1} + \cdots + d_{l_p i^*}) \\ &\geq f(l_p, u_j^{(m)} + d_{jl_1} + \cdots + d_{l_{p-1} l_p}) \\ &\vdots \\ &\geq f(j, u_j^{(m)}) \end{aligned}$$

This contradicts (3.17).

Once a node i^* is selected by (3.17) and (3.14) is calculated for k, $(i^*, k) \in A$, we do not need transportations (3.14) from node i^* any more. For example, we may take the next function as $f(\cdot, \cdot)$.

(3.21) $f(i,t) = t + \mu(i,n)$

where $\mu(i,n)$ is the shortest path length from node i to node n without taking account of time-window constraints. This is given by the Yen's method in computational complexity $O(n^3/4)$.

Summarizing the above arguments, we propose an algorithm, which is different from two other methods, namely the B-F-like method and the Yen-like method. In what follows, E denotes the set of nodes selected by (3.17).

(i) Calculate $\mu(i,n)$ for $i \in N$ using the Yen's method[9], and set

(3.22) $m = 0, \quad u_1^{(0)} = t_0, \quad u_i^{(0)} = \infty \ (i \neq 1), \quad E = \emptyset, \quad I = \{1\}.$

(ii) If $I - E = \emptyset$, then stop. The problem is infeasible.

Otherwise, find a node i^* satisfying the following for the function f of (3.21).

(3.23) $f(i^*, u_{i^*}^{(m)}) = \min_{j \in I-E} f(j, u_j^{(m)}).$

If $i^* = n$, then stop. The shortest path has been obtained.

Otherwise, revise $u_k^{(m)}$ by (3.14) for all nodes k, $(i^*, k) \in A$, $k \notin E$. If the revision changes $u_k^{(m)}$ from ∞ to a finite value, add node k to I.

(iii) Let $E = E \cup \{i^*\}$ and $m = m + 1$. Go back to (ii).

In the above algorithm, the revision of $u_k^{(m)}$ and the evaluation (3.23) by the function f can be regarded respectively as the branching procedure and the bounding procedures of the branch and bound method. For this reason, we name the algorithm the BAB-like method. The computational complexity of this method is evaluated as $O(m_0 n^2 + n^3/4)$. This algorithm terminates when $i^* = n$ occurs in (ii), at this moment all the earliest arrival times at every nodes are not necessarily obtained. This makes the BAB-like method more efficient than other methods.

4. Numerical Example

In the previous section, we elucidated the computation complexity of three methods, theoretically. In this section, we evaluate the practical performances of them by some examples.

We apply the proposed methods to various networks, in which not only the number of nodes or arcs but also the number of open time periods in a time-window vary, and measure CPU-time of a computer. Hitachi main-frame computer S-3600/120A and FORTRAN language were used. We change the number of nodes from 20 to 100 by 20. For each number of nodes, say n, the number of directed arcs, say m, are determined according to three types of formulas; $n \times 5$, $n \times 10$ and $n \times 15$. That is, three formulas make a nodes have 5, 10 and 15 directed arcs, respectively. The number of open time periods in a node's time-window, say p, changes from 2 to 6 by 2 and the same number of open time periods are given on a directed arc's time-window. For given numbers of nodes, arcs and open time periods, we generate a random network with time-windows as follows.

(i) In a square having area $L \times L$, n points are located randomly and assigned to n nodes.

(ii) For each node, $n - 1$ Euclidean distances between this node and all other nodes are sorted in the decreasing order. Arcs with the 5-, 10- or 15-th shortest distances are adopted according to formulas $n \times 5$, $n \times 10$ and $n \times 15$, respectively. The lengths of arcs are the same as their distances.

(iii) For the given number of open time periods, say p periods, $2 \times (p - 1)$ time points are picked up randomly in $[0, \sqrt{L^2 + L^2}]$, where $\sqrt{L^2 + L^2}$ corresponds to the length of the diagonal of the square. Denoting these time points by $t_1 < t_2 < \cdots < t_{2(p-1)}$, $[0, t_1], [t_2, t_3], \cdots, [t_{2(p-1)}, \infty]$ are adopted as p open time periods.

For the network constructed by the above procedure, we make an AGV-type SPPTW by selecting a start node and a destination node randomly, and setting starting time $t_0 = 0$. About the above time-windows construction (iii), we should note that open time periods before starting time point $t_0 = 0$ are of no use and time-windows consisting of only finite time points makes the SPPTW to be likely infeasible. This is the reason that the time-window is generated so as to include only positive time points and an infinite time point in the last open time period.

We generated 10 random networks for one case of the given number of nodes, arcs and open time periods, and provided 10 SPPTWs for a network by selecting a pair of the start node and the destination node 10 times. In consequence, we solve 100 SPPTWs for a case of the given number of nodes, arcs and open time periods setting $L = 100$, and measured mean values and standard deviations of CPU-time. In Table 1-a, we compare the performance of three methods, B-F-like, Yen-like and BAB-like methods for the case of 2 open time periods' time-windows. Figures represent the mean CPU-time while figures in parentheses do the standard deviation.

As seen by Table 1-a and the theoretical discussion in the previous section, it is clear that the Yen-like method is always superior to the B-F-like one from the point of computational efficiency. Therefore, the computational results about the B-F-like method are omitted to be displayed and discussed hereafter. For the case of 4 and 6 open time periods, the results were obtained in Table 1-b and -c.

Table 1-a. CPU-time(msec) for the case of 2 open time periods

No. of arcs	Methods	No. of nodes (n)				
		20	40	60	80	100
$5 \times n$	B-F-like	4(1)	9(1)	16(2)	20(2)	29(7)
	Yen-like	3(0.5)	7(1)	12(2)	16(2)	23(3)
	BAB-like	1(0.5)	2(1)	3(1)	4(1)	6(1)
$10 \times n$	B-F-like	6(1)	18(2)	29(5)	39(7)	57(11)
	Yen-like	6(0.5)	14(1)	22(3)	33(5)	43(5)
	BAB-like	1(0.5)	3(1)	5(2)	7(3)	9(4)
$15 \times n$	B-F-like	8(1)	20(3)	36(4)	54(8)	69(9)
	Yen-like	7(1)	19(3)	28(4)	41(6)	58(7)
	BAB-like	1(1)	4(1)	6(2)	8(3)	13(6)

Table 1-b. CPU-time(msec) for the case of 4 open time periods

No. of arcs	Methods	No. of nodes (n)				
		20	40	60	80	100
$5 \times n$	Yen-like	5(1)	12(3)	20(4)	28(4)	38(5)
	BAB-like	1(0.5)	2(1)	3(1)	5(2)	8(3)
$10 \times n$	Yen-like	9(1)	21(3)	40(6)	53(7)	72(7)
	BAB-like	1(0.5)	3(1)	5(2)	8(5)	11(6)
$15 \times n$	Yen-like	13(1)	29(3)	46(6)	69(9)	96(11)
	BAB-like	1(0.5)	4(2)	6(3)	9(5)	11(5)

Table 1-c. CPU-time(msec) for the case of 6 open time periods

No. of arcs	Methods	No. of nodes (n)				
		20	40	60	80	100
$5 \times n$	Yen-like	7(1)	17(4)	26(5)	39(5)	49(5)
	BAB-like	1(0.5)	2(1)	4(2)	7(4)	9(5)
$10 \times n$	Yen-like	13(2)	30(4)	48(3)	73(9)	99(9)
	BAB-like	1(0.5)	3(2)	5(3)	8(4)	11(7)
$15 \times n$	Yen-like	17(2)	40(4)	60(8)	97(11)	133(17)
	BAB-like	1(0.5)	4(2)	6(2)	9(5)	12(7)

From Table 1, we obtain some remarks.

(1) For two method, the Yen-like and BAB-like method, their CPU-time grow up by the proportional rate between the 1-st power and 2-nd power of the number of nodes n.

This rate increases by the number of arcs m. For the Yen-like method, the increase of the rate by the number of open time periods p, can be recognized too. However, it is not true for the BAB-like method. The increase of p hardly produces any effect on CPU-time for the BAB-like method.

(2) With keeping n and p constant, CPU-time increases with the proportional rate below the 1-st power of m for both methods. In detail, the rate of the BAB-like method is lower than one of the Yen-like method.

(3) The increase of p raises CPU-time for the Yen-like method but not for the BAB-like method.

(4) The standard deviation of CPU-time synchronizes with the mean value. In the same condition, it is larger for the Yen-like method than for the BAB-like method. Therefore, the BAB-like method has more constant efficiency than the Yen-like method.

In consequence, we may conclude that the BAB-like method is most preferable in all aspects, which has been estimated by the theoretical evaluation of the computational complexity in the previous section. The theoretical evaluation is executed for the worst case. As known in the above remarks, the practical efficiency by numerical examples indicates more better result than the theoretical evaluation. The superiority of the BAB-like method comes from the fact that the BAB-like method terminates its algorithm as soon as the arrival time on the destination node is recognized to be optimal, which can be easily checked by using the classical shortest distance without considering any time-windows.

5. Concluding Remarks

In this paper, the shortest path problem on the network with AGV-type time-windows is investigated. The problem can be decomposed into subproblems, each of which has the similar structure to the classical shortest path problem with no time-window. Due to this fact, algorithms similar to those of Bellman-Ford and Yen are developed. Another algorithm, called the BAB-like method, is also proposed and it is clarified that it is the most efficient method for the computational complexity, theoretically and practically. Originally, the problem is motivated by the routing problem of the automated guided vehicle in manufacturing system. Furthermore, this problem and the proposed algorithms are able to be applied to other fields, such as the transportation system, the project scheduling and so on.

References

[1] Bellman, R.E. : On a Routing Problem. *Quart. Applied Math.*, Vol.**16**(1958), 87–90.

[2] Desrosiers, J., Pelletier, P. and Soumis, F. : Plus Court Chemin avec Contraintes d'horaires. *R.A.I.R.O Recherche Operationnelle*, Vol.**17**(1983), 357–377.

[3] Desrosiers, J., Soumis, F. and Desrochers, M. : Routing with Time Windows by Column Generation. *Networks*, Vol.**14**(1984), 545–565.

[4] Desrochers, M. and Soumis, F. : A Generalized Permanent Labelling Algorithm for the Shortest Path Problem with Time Windows. *Information Systems and Operational Research*, Vol.**26**(1988), 191–212.

[5] Ford, Jr., L.R. : Network Flow Theory. *The RAND Corp.*, **P-923**(1956).

[6] Fujii, S., Sandoh, H. and Hohzaki, R. : Routing Control of Automated Guided Vehicles in FMS. *Proc. of 1988 U.S.A.-Japan Symposium on Flexible Automation*, Vol.**1**(1988), 629–636.

[7] Hohzaki, R., Fujii, S. and Sandoh, H. : A Routing Method of Automated Guided Vehicles in FMS by the Time-Windows Constrained Shortest Path. *Proc. of 1990 Japan-*

U.S.A. Symposium on Flexible Automation, Vol.**2**(1990), 485–492.

[8] Solomon, M. and Desrosiers, J. : Time Window Constrained Routing and Scheduling Problems. *Transportation Science*, Vol.**22**(1988), 1–13.

[9] Yen, J.R. : An Algorithm for Finding Shortest Routes from All Source Nodes to a Given Destination in General Network. *Quart. Applied Math.*, Vol.**27**(1970), 526–530.

Appendix : Proof of Theorem 1

From equation (2.11), we obtain

$$T_{ik} \otimes (T_{jl} \ominus d_{ij}) \neq \emptyset. \qquad (A.1)$$

Therefore, it suffices to prove that the above inequality is valid for at most $n_i + n_j - 1$ combinations of k and l.

$T(i) = \{T_{ik};\ k = 1, \cdots, n_i\}$ and $T(j) \ominus d_{ij} = \{T_{jl} \ominus d_{ij}\}$ consist of n_i and n_j intervals, respectively. Now, assume there are the set of m intervals and the set of n intervals. Let $K[m,n]$ be the number of intervals which the product set of these sets contains. The following inequality is to be proved.

$$K[m,n] \leq m + n - 1 \qquad (A.2)$$

In the case of $m = 1$, $n = 1$, (A.2) is clearly valid. Assume that (A.2) is valid in the case of $m \leq M$ and $n \leq N$. $K[M, N+1]$ becomes the maximum in such as Fig.1.

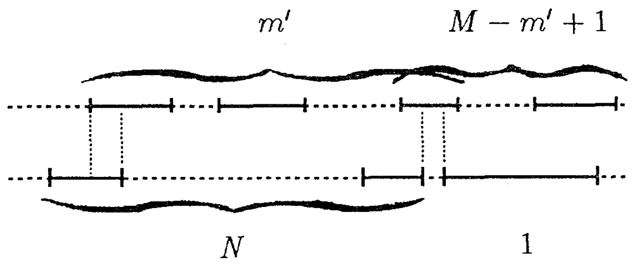

Fig.1 Product set of intervals

Therefore, we have

$$\begin{aligned} K[M, N+1] &\leq \max_{1 \leq m' \leq M} \{K[m', N] + K[M - m' + 1, 1]\} \\ &\leq \max_{1 \leq m' \leq M} \{(N + m' - 1) + (M - m' + 1)\} \\ &= M + (N+1) - 1. \end{aligned}$$

Likewise, $K[M+1, N] \leq (M+1) + N - 1$. In consequence, (A.2) is valid for arbitrary integers m and n.

Ryusuke HOHZAKI:
Department of Applied Physics,
National Defense Academy,
1-10-20 Hashirimizu, Yokosuka,
Kanagawa 239, Japan
E-mail: hozaki@cc.nda.ac.jp