Journal of the Operational Research Society (2003) 54, 860–870
www.palgrave-journals.com/jors

Time analysis for planning a path in a time-window network

Y-L Chen[1], L-J Hsiao[2] and K Tang[3,*]

[1]*National Central University, Chung-Li, Taiwan, Republic of China;* [2]*Van-Nung Institute of Technology, Chung-Li, Taiwan, Republic of China; and* [3]*Purdue University, West Lafayette, IN, USA*

A systematic method is proposed to generate time information on the paths and nodes on a time-window network for planning and selecting a path under a constraint on the latest entering time at the destination node. Specifically, three algorithms are proposed to generate six basic time characteristics of the nodes, including the earliest and latest times of arriving at, entering, and departing from each node on the network. Using the basic time characteristics, we identify inaccessible nodes that cannot be included in a feasible path and evaluate the accessible nodes' flexibilities in the waiting time and staying time. We also propose a method for measuring adverse effects of including an arc. Finally, based on the time characteristics and the proposed analyses, we develop an algorithm that can find the most flexible path in a time-window network.
Journal of the Operational Research Society (2003) **54,** 860–870. doi:10.1057/palgrave.jors.2601583

Keywords: network; time-window constraint; time analysis

Introduction

The time-constrained network analysis generalizes the traditional network analysis by incorporating the constraints on the availability of the nodes over time. Time window has been a common form of time constraint considered in the literature. Basically, a time window is a time period, defined by the earliest and latest times, when a node is available for traveling through. There are many practical situations where time windows can be used to describe the time constraints associated with the nodes and arcs on a network. For example, a time window in a transportation network may be the time period that a service or transition facility is available for the traveler to pass through. Similarly, a time window associated with an arc may be the time that a transportation channel is open. In vehicle routing problems and traveling salesman problem, a time window may represent the time period during which a customer needs to be visited, and in production scheduling problems, a time window may be the time that a certain machine or production facility is available for processing jobs.

The time-window network analysis has attracted widespread interest over the past decade, and many fundamental issues considered in the traditional network optimization have been addressed for time-window networks. These include the vehicle routing problem, the traveling salesman problem, and the shortest path problem.

*Correspondence: K Tang, Krannert Graduate School of Management, Purdue University, West Lafayette, IN 47907-1310, USA.
E-mail: ktang@mgmt.purdue.edu*

As in other operations research studies, mathematical models are used in network analysis to provide an abstract representation of the real problems for assessing decision alternatives and for selecting the best solution. Because of the complexity of a real problem and the tractability of the model, an exact representation of the real problem is seldom the objective of a model formulation effort. For example, the waiting time at a node or arc may be a very important factor in path selection because of the cost, safety, quality, and other consideration. It is difficult, however, to incorporate the factors of this nature in a network model. In fact, adding a simple cost component as a linear function of waiting time, a network model may become very complicated to solve. Consequently, many preferences, tangible and intangible factors are not incorporated in model formulation, resulting in implementation issues.

In this paper, instead of determining an 'optimal' path, our objective is to develop a systematic method to generating important time-related information for each node and arc on a network to assist selecting the most flexible path. We consider a time-window network and assume that a deadline constraint is imposed on the latest entering time of the destination node. Note that there could be multiple feasible paths under a deadline constraint. In planning or selecting a path, the following questions are often considered:

(a) What is the latest time to depart from a node on the network without violating the deadline constraint?
(b) What is the waiting time before entering a node? Can the waiting time be reduced and how much of it is inevitable?

(c) How long can we stay at a node? Can we increase or reduce the staying time, and what are the minimum and maximum staying times at a node?

The answers to these questions provide valuable information for selecting and planning a path on the network. For example, the waiting time before entering a node and the staying time at a node may have different economic or risk implications, depending on the application. A preferred path can be selected from the feasible paths by evaluating alternative paths based on the time information from our proposed analysis. We can also obtain information on the flexibility of a path in terms of the 'spare time' associated with the nodes on the path. A more flexible path is desirable if we consider potentially unexpected delays in traveling on the network. Furthermore, the information is also helpful when we have to switch to a new path after we have finished a portion of the initial path, because a node or more on the original path become unreliable.

Specifically, we propose three algorithms to generate six basic time characteristics associated with the nodes on the network. Using the time characteristics, we first identify inaccessible nodes, the nodes that cannot be included in a feasible path, and propose further analyses on the waiting time and staying time for accessible nodes. From the analyses, we evaluate the flexibility associated with each accessible node in terms of its flexible times in waiting and staying. We also identify inflexible nodes, which have no spare time in entering, departure, or both. It is demonstrated that the time characteristics and analyses provide valuable information for selecting and planning a path.

The remainder of this paper is organized as follows. First, we give the problem statement, and present the algorithms for evaluating six basic time characteristics associated with the nodes on the network. In the subsequent section, we use the basic time characteristics to further analyze the waiting time and the staying time at the nodes, and additional properties associated with the nodes and arcs. Using the results from the analyses, we then discuss a method of selecting a path in practice. A conclusion is given in the final section.

Problem statement and algorithms

A time-window network is represented as a directed graph $G=(N, A)$, where $N=\{1, 2, 3, \ldots, n\}$ and $A=\{(i, j)|i, j\in N$, and $i\neq j\}$. For simplicity, we assume that nodes 1 and n are the source and the destination, respectively. For each arc $(i, j)\in A$, there is a static and non-negative time duration, denoted by $dur(i, j)$, to pass through it.

For each node $i\in N$, there are two types of time-window constraints, entering-time and departure-time windows. Let E_i be the number of entering-time windows for node i. The kth entering-time window for node i, $1\leq k\leq E_i$, is represented by $[Ebegin_i^k, Eend_i^k]$, where $Ebegin_i^k$, and $Eend_i^k$ are the beginning and ending times of the time window, respectively. Similarly, $(Dbegin_i^k, Dend_i^k)$ is used to represent the kth departure-time window, $1\leq k\leq D_i$, for node i.

We assume that all the entering-time windows of a node are disjoint, that is, for node i,

$$Eend_i^{k-1} < Ebegin_i^k < Eend_i^k \text{ for } 1<k\leq E_i$$
$$\text{and } 0\leq Ebegin_i^1\leq Eend_i^1$$

and, similarly, the departure-time windows for a node are also disjoint:

$$Dend_i^{k-1} < Dbegin_i^k \leq Dend_i^k \text{ for } 1<k\leq D_i$$
$$\text{and } 0\leq Dbegin_i^1\leq Dend_i^1$$

Furthermore, for simplicity in designing and presenting the algorithms, we let the source node have only one entering-time window $[0, \infty]$ and the destination node have only one departure-time window $(0, \infty)$. Without loss of generality, we let the ending time of the last entering-time window of node n, $Eend_n^{E_n}$, be the predetermined latest time of entering the destination.

For illustration, let us consider the time-window network in Figure 1, where nodes 1 and 12 are the source and destination nodes, respectively, and the deadline to enter node 12 is 33. The number on an arc is the arc's duration time, the time intervals given by the brackets are entering-time windows, and those given by the parentheses are departure-time windows. For example, node 9 has one entering window and one departure window.

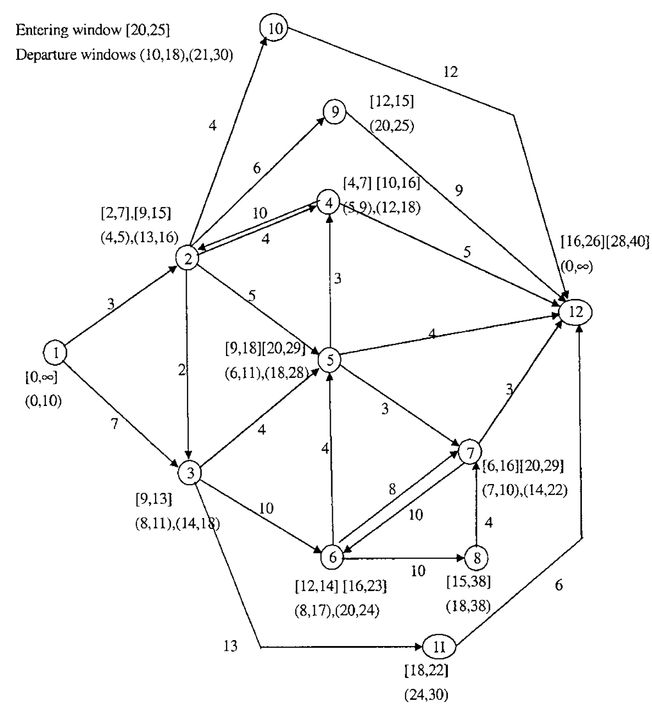

Figure 1 An example of time-window network.

In order to generate useful time information for path selection, we define the following six basic time characteristics associated with each node:

EA_i = earliest time to arrive at node i from the source node,
EE_i = earliest time to enter node i,
ED_i = earliest time to depart from node i after entering,
LD_i = latest time to depart from node i without violating the deadline constraint,
LE_i = latest time to enter node i without violating the deadline constraint,
LA_i = latest time to arrive at node i without violating the deadline constraint.

There are several apparent relations among these time characteristics and the time-window constraints. First, the earliest time to enter node i must be no earlier than the earliest time to arrive at node i ($EE_i \geq EA_i$), and EE_i must be contained in an entering-time window associated with node i. Second, we have $ED_i \geq EE_i$, and ED_i must be contained in a departure-time window associated with node i. Similarly, the same kind of relations holds for the latest times LD_i, LE_i, and LA_i. It is evident that the time characteristics associated with a node on a path should satisfy the relations $LA_i \geq EA_i$, $LE_i \geq EE_i$, and $LD_i \geq ED_i$. If the time characteristics associated with a node violate any one of these relations, a path containing the node is not feasible, and we define a node of that nature as an *inaccessible* node.

Next, we present three algorithms for evaluating the six time characteristics defined above. The three algorithms are executed in sequence. In presenting the algorithms, we define that a node, say node j, is a predecessor of node i if arc (j, i) exists on the network. Similarly, node j is a successor of node i if arc (i, j) exists.

The first algorithm is a forward procedure which evaluates EA_i, EE_i, and ED_i for each node i on the network from the source to the destination. Let $EA(j, i)$ be the earliest time to arrive at node i through arc (j, i). Then, we have $EA(j, i) = ED_j + dur(j, i)$, and EA_i for node i is determined by

$$EA_i = \underset{j}{Min}\{EA(j, i)\}$$

for all the predecessor nodes j of i. Once EA_i is obtained, EE_i equals to EA_i if EA_i falls in one of the entering-time windows of node i; otherwise EE_i equals to $Ebegin_i^k$ of the next available entering-time window. Finally, ED_i is obtained by comparing EE_i with the available departure-time windows at node i.

The second algorithm is a backward procedure for evaluating LE_i and LD_i from the destination to the source. Let $LD(i, j)$ denote the latest time to leave nodes i to j without violating the deadline constraint, which can be obtained by

(i) Subtracting $dur(i, j)$ from LE_j.
(ii) Adjusting the value to the departure-time windows of node i.

In turn, LD_i can be determined as $\text{Max}_j\ \{LD(i, j)\}$, for all the successor nodes j of node i. Finally, LE_i is determined by adjusting LD_i to its entering-time windows.

The third algorithm is used to evaluate LA_i. Let $LA(j, i)$ be the latest time to arrive at node i from node j without violating the deadline constraint, which can be obtained by the following procedure:

(i) Let $x = LE_i - dur(j, i)$.
(ii) Let y denote the latest time that we can leave node j for node i.

If $x \geq LD_j$, then $y = LD_j$
Else if $ED_j \leq x \leq LD_j$, then determine y by comparing x and the departure-time windows of node j
Else if $x \leq ED_j$, then $y = -\infty$.

(iii) Set $LA(j, i) = dur(j, i) + y$.

Finally, LA_i can be determined as $\text{Max}_j\ \{LA(j, i)\}$ for all the predecessor nodes j of i.

The proofs of the algorithms are omitted because the first and second algorithms are simple modifications of the classical shortest path algorithm and the third algorithm is intuitive. For the detailed algorithms, see the appendix.

We continue using the network in Figure 1 for illustration. We first use Algorithm I to obtain the EA_i, EE_i, and ED_i for each node. The algorithm begins with setting $ED_1 = 0$ for the source node. Since the successors of the source node are nodes 2 and 3, the next step is to modify the EA_j, EE_j, and ED_j of these two nodes. For node 2, we find that $EA_2 = ED_1 + dur(1, 2) = 0 + 3 = 3$. Since EA_2 is in the first entering-time window, [2,7], $EE_2 = EA_2 = 3$. We also find $ED_2 = 4$ because the first departure-time window (4, 5), is behind EE_2. For node 3, the computation can be down similarly. Repeatedly using the procedure, we obtain the values of EA_i, EE_i, and ED_i of all the nodes given in Table 1.

Since Algorithm II is a backward procedure, we first set the value of LE_{12} to be $Eend_j^{E_n}(=33)$. Node 12 has six predecessors: nodes 4, 5, 7, 9, 10, and 11. Consider node 4 first. The difference between LE_{12} and $dur(4, 12)$ is 28, which is the latest departure time without considering departure-time windows. However, since the last departure-time window of node 4 is (12, 18), the latest departure time at node 4, LD_4, is 18. By matching LD_4 with the entering-time windows, we found that the latest time to enter node 4 (LE_4) is 16. Using the procedure, we found that $LD_5 = 28$, $LE_5 = 28$, $LD_7 = 22$, $LE_7 = 22$, $LD_9 = 24$, $LE_9 = 15$, $LD_{10} = 21$, $LE_{10} = 21$, $LD_{11} = 27$, and $LE_{11} = 22$. In the next iteration, node 5 is considered and can be done similarly. The complete results from Algorithm 2 are listed in the columns under LE_i and LD_i in Table 1.

In Algorithm III, all the LA_i values can be computed independently and in any order since their evaluation processes are not dependent on the LA_i values of other nodes. To illustrate this point, we give the process of obtaining LA_5. Node 5 has three predecessors, namely,

Table 1 Time characteristics associated with the nodes in Figure 1 when the deadline is 33

Node i	EA_i	EE_i	ED_i	LA_i	LE_i	LD_i
1	0 (source)	0 (source)	0	— (source)	10	10 (into node 2)
2	3 (from node 1)	3	4	15 (from node 4)	15	16 (into node 5)
3	6 (from node 2)	9	9	13 (from node 1)	13	18 (into node 5)
4	8 (from node 2)	10	12	14 (from node 5)	16	18 (into node 12)
5	9 (from node 2)	9	9	28 (from node 6)	28	28 (into node 12)
6	19 (from node 3)	19	20	21 (from node 3)	23	24 (into node 5)
7	12 (from node 5)	12	14	22 (from node 5)	22	22 (into node 12)
8	30 (from node 6)	30	30	$-\infty$	18	18 (into node 7)
9	10 (from node 2)	12	20	11 (from node 2)	15	24 (into node 12)
10	8 (from node 2)	20	21	20 (from node 2)	21	21 (into node 12)
11	22 (from node 3)	22	24	22 (from node 3)	22	27 (into node 12)
12	13 (from node 5)	16	16	33 (from node 9)	33	— (destination)

Inaccessible node: node 8.
Inaccessible arcs: (6, 7), (7, 6), (8, 6), (6, 8) and (4, 2).
Inflexible node in arrival time: node 11.
Inflexible node in departure time: node 10.

nodes 2, 3, and 6. Consider node 2 first ($j=2$ and $i=5$). We obtain that $x=LE_5-dur(2, 5)=28-5=23$ and y, the latest time to depart from nodes 2 to 5, equals 16. Consequently, we obtain $LA(2, 5)=y+dur(2, 5)=16+5=21$. Next, we consider node 3($j=3$ and $i=5$), and we obtain $x=LE_5$-$dur(3, 5)=28–4=24$ and $y=18$. Thus, our result becomes $LA(3, 5)=y+dur(3, 5)=22$. Finally, we consider node 6, and the result is $x=24$, $y=24$, and $LA(6, 5)=y+dur(6, 5)=28$. Using the results associated with these three nodes, we obtain $LA_5=\max\{LA(2, 5), LA(3, 5), LA(6, 5)\}=28$. The results for the remaining nodes are given in Table 1.

Time analysis

In the last section, three algorithms are developed to evaluate six basic time characteristics for all the nodes on the network. In this section, we first define a partial order precedence graph to summarize the relations among the time characteristics. Next, we identify inaccessible nodes and propose further analyses on the waiting time and staying time for accessible nodes. Based on the analyses, we evaluate the flexibility associated with each accessible node in terms of its flexible times in waiting and staying. We also identify inflexible nodes, which have no spare time in entering time, departure time, or both. It is demonstrated that the time characteristics and the proposed analyses provide valuable information for planning a path in a time-window network.

First, we use AA_i, AE_i, and AD_i to denote the actual arriving time, entering time, and departure time for node i, respectively. For a feasible path, the following relations must hold:

$$EA_i \leq AA_i \leq LA_i, \quad EE_i \leq AE_i \leq LE_i \quad \text{and} \quad ED_i \leq AD_i \leq LD_i$$

Furthermore, the arrival times, the entering times, and the departure times have the following relations:

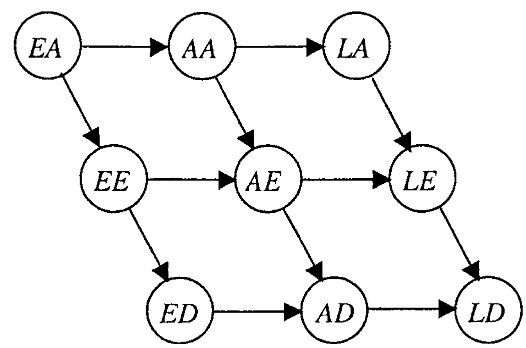

Figure 2 Partial order relations among the time characteristics.

$$EA_i \leq EE_i \leq ED_i, \quad AA_i \leq AE_i \leq AD_i \quad \text{and} \quad LA_i \leq LE_i \leq LD_i$$

Using these two sets of relations, we obtain a partial order precedence graph shown in Figure 2 to summarize the relations among the time characteristics.

Inaccessible nodes

As defined, the inaccessible nodes are the nodes of which at least one of the inequalities, $EA_i \leq LA_i$, $EE_i \leq LE_i$, or $ED_i \leq LD_i$, is not satisfied. For example, from Table 1, we find that node 8 is an inaccessible node. The inaccessible nodes should not be included in a path unless the latest time to enter the destination or the time-windows associated with the node can be changed. In the following analysis, all the inaccessible nodes are excluded.

Node flexibility

We define the arrival-time interval for node i as $[EA_i, LA_i]$, which gives limits on the actual arrival time of a feasible path. Similarly, we define $[EE_i, LE_i]$ and $[ED_i, LD_i]$ as the entering-time and departure-time intervals for node i, respectively. The ranges of the three time intervals are defined as follows:

$$Range_{\mathrm{A}}[i] = LA_i - EA_i,\ Range_{\mathrm{E}}[i] = LE_i - EE_i,$$
$$Range_{\mathrm{D}}[i] = LD_i - ED_i$$

A larger range associated with a node implies more flexibility at a node or more robustness of a path if we consider uncertainties such as a possible unexpected increase in the duration time before arriving at the node. For a given network, the three intervals may be overlapped or disjoint.

If the earliest arrival time at a node is identical to its latest arrival time ($Range_{\mathrm{A}}[i]=0$), we would not have any flexibility in the arrival time at the node. In other words, any delay of arriving at the node results in failing to meet the given latest time of entering the destination. In this case, we say that the node is inflexible in arrival time. Furthermore, if the earliest departure time is identical to the latest departure time ($Range_{\mathrm{D}}[i]=0$), the node is inflexible in departure time. If a node is not flexible in both arrival time and departure time, we call the node an inflexible node.

Waiting time analysis

We consider the waiting time between arriving at a node and entering the node. There are two situations. In the first situation, the arrival-time interval overlaps with the entering-time interval, that is, $LA_i \geq EE_i$. In this situation, the waiting time can be zero, if the arrival time at the node is in the overlapped period $[EE_i, LA_i]$ of the arrival-time and the entering-time intervals. On the other hand, if the actual arrival time AA_i is before time EE_i, then we have to wait to enter the node during the time period $[AA_i, EE_i]$.

In the second situation, the arrival-time and the entering-time intervals are disjoint, that is, $LA_i < EE_i$. In this situation, we have an unavoidable waiting time interval from LA_i to EE_i. As shown in Figure 3, the longest possible waiting time is given by

$$longest - waiting[i] = LE_i - EA_i$$

and the total actual waiting time, AE_i–AA_i, can be partitioned into three portions following the relations among the latest arrival and the earliest entering times. These three times are defined as follows:

$$shortest - waiting[i] = EE_i - LA_i,$$
$$prewaiting[i] = LA_i - AA_i,$$
$$postwaiting[i] = AE_i - EE_i$$

Note that in the first situation where the arrival- and entering-time intervals overlap, we only have pre-waiting time and post-waiting time. In the second situation where $LA_i < EE_i$, the interval between LA_i and EE_i is the shortest-waiting interval because the interval is unavoidable.

On the contrary, the pre-waiting time and post-waiting time are flexible because they can be expanded or reduced by

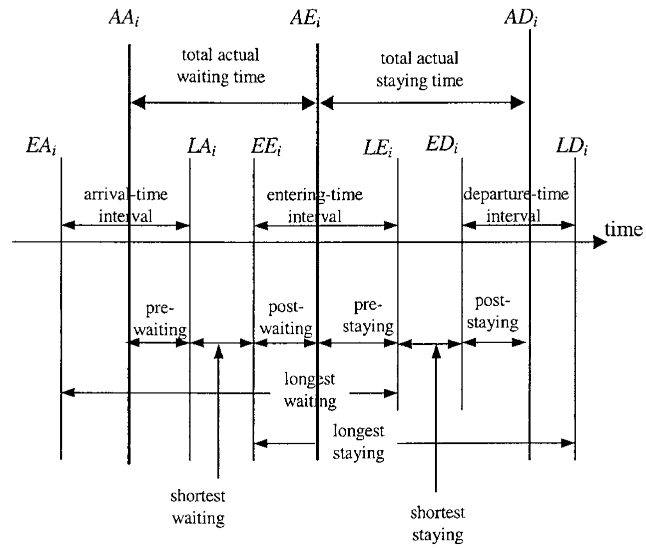

Figure 3 Time ranges and their relations.

changing the arrival or entering times. Since AA_i falls in the arrival-time interval and AE_i in the entering-time interval, we have the relations $0 \leq$ pre-waiting$[i] \leq LA_i$–EA_i and $0 \leq$ post-waiting$[i] \leq LE_i$-EE_i. Therefore, the total flexible waiting time in both situations can be measured by $(LA_i–EA_i) + (LE_i-EE_i) + \min\{shortest\text{-}waiting[i], 0\}$.

Staying time analysis

We can analyze the staying time using the method used for analyzing the waiting time. There are two situations, based on whether the entering-time interval overlaps with the departure-time interval. In the first situation, the entering-time interval overlaps with the departure-time interval, that is, $LE_i \geq ED_i$. In this situation, the staying time can be zero, if the entering time at the node is in the overlapped period, $[ED_i, LE_i]$, of the entering-time and the departure-time intervals. On the other hand, if the actual entering time AE_i is before time ED_i, then we have to stay at the node during the time period $[AE_i, ED_i]$. In the second situation, the entering-time and the departure-time intervals are disjoint, that is, $LE_i < ED_i$. In this situation, we have an unavoidable staying time interval from LE_i to ED_i.

The time of actually staying at node i, denoted by *actual-staying*$[i]$, is determined by the difference between AD_i and AE_i. The actual staying time consists of three portions: the pre-staying time, shortest-staying time, and post-staying time, defined by

$$prestaying[i] = LE_i - AE_i,$$
$$shorteststaying[i] = ED_i - LE_i,$$
$$poststaying[i] = AD_i - ED_i$$

The shortest staying time at node i is determined by max $\{shortest\text{-}staying[i], 0\}$, which is the shortest time of staying at the node after entering. Next, we discuss the flexible staying time, which includes the pre-staying time and post-staying time. Basically, the flexible staying time may be expanded or reduced within our discretion. We can determine the length of the pre-staying time by changing the actual entering time and that of the post-staying time by our actual departure time. Since AE_i and AD_i fall within the entering-time interval and the departure-time interval, respectively, we have $0 \leq \text{pre-staying}[i] \leq LE_i - EE_i$ and $0 \leq \text{post-staying}[i] \leq LD_i - ED_i$. Therefore, the total flexible staying time can be represented as

$$(LE_i - EE_i) + (LD_i - ED_i) + \min\{\textit{shortest-staying}[i], 0\}$$

The *longest-staying*[*i*] is the maximal staying time at a node. It is the longest time that we may stay at a node, which includes the entering-time interval, shortest staying time, and departure-time interval. The relation is

$$\begin{aligned} \textit{Longest-staying}[i] &= LD_i - EE_i \\ &= \textit{Range}_{\mathrm{E}}[i] + \textit{shortest-staying}[i] + \textit{Range}_{\mathrm{D}}[i] \end{aligned}$$

Adverse effects by including an arc

It is often desirable to analyze potential adverse effects if we consider including arc (i, j) into the path. If arc (i, j) is included, the earliest time to arrive at node j becomes $EA(i, j)$, the latest time to arrive at node j without violating the deadline constraint becomes $LA(i, j)$, and the latest time to leave node i is $LD(i, j)$. As for the earliest time to depart from node i, it is the same as the original ED_i because at time ED_i we are allowed to leave for any successor nodes of node i.

There are three types of adverse effects that may be incurred by including arc (i, j) in the path. The first is associated with node i. Since $LD(i, j) \leq LD_i$, the new departure-time interval of node i, $[ED_i, LD(i, j)]$, is smaller than the original one $[ED_i, LD_i]$, implying less flexibility in the departure time from the node.

The second type of adverse effects is associated with node j. Since $EA(i, j) \geq EA_j$ and $LA(i, j) \leq LA_j$, the original arrival-time interval of node j, $[EA_j, LA_j]$, is reduced to a smaller interval $[EA(i, j), LA(i, j)]$, implying a less flexibility in the arrival time of the node. Furthermore, it may also increase the shortest waiting time. When $LA_j < EE_j$, the shortest waiting time interval at node j will be increased from $[LA_j, EE_j]$ to $[LA(i, j), EE_j]$.

The third type of effects is the possibility of causing the path to be infeasible. In this case, we may call arc (i, j) an inaccessible arc, which can be easily identified by checking the condition $LA(i, j) < EA(i, j)$. When the condition is satisfied, it is an inaccessible arc because the earliest time to arrive at node j is later than the latest time we need to arrive at node j without violating the deadline constraint.

Example 1

Using the basic time characteristics in Table 1 derived for the network of Figure 1, we obtain the additional properties associated with arrival, entering, and departure times of the nodes. For example, consider related time intervals for nodes 9 and 4 given in Table 2. The arrival-, entering-, and departure-time intervals of node 9 are disjoint, suggesting that no matter how the arrival, entering, and departure times are selected, we cannot completely eliminate waiting and staying periods. Furthermore, since the three intervals of node 4 overlap, all the waiting and staying time intervals are flexible and can be adjusted by selecting proper arrival, entering, departure times.

Table 2 Results of time analysis for nodes 4 and 9

Result	*Node 4*	*Node 9*
Arrival-time interval	[8, 14]	[10, 11]
Entering-time interval	[10, 16]	[12, 15]
Departure-time interval	[12, 18]	[20, 24]
Pre-waiting interval	$[AA_4, 14]$	$[AA_9, 11]$
Shortest-waiting interval	NA	[11, 12]
Post-waiting interval	$[10, AE_4]$	$[12, AE_9]$
Pre-staying interval	$[AE_4, 16]$	$[AE_9, 15]$
Shortest-staying interval	NA	[15, 20]
Post-staying interval	$[12, AE_4]$	$[20, AD_9]$
Flexible waiting time	8	4
Fixed waiting time	0	1
Longest waiting time	8	5
Flexible staying time	8	7
Fixed staying time	0	5
Longest staying time	8	12

Path selection

In this section, we propose an algorithm for selecting a path with maximum flexibility to meet the deadline requirement at the destination. As discussed in the last section, $\textit{Range}_{\mathrm{A}}[i]$, $\textit{Range}_{\mathrm{E}}[i]$, and $\textit{Range}_{\mathrm{D}}[i]$ are reasonable measurements for a node's flexibility in its arrival, entering, and departure times. We first define a *flexible* node, based on a weighted sum of these ranges. Using the definition, we define that a path is flexible if all the nodes on the path are flexible. Under these definitions, an algorithm is proposed for finding the path with maximum flexibility among all flexible paths from the source node to the destination.

We define a measurement for the overall flexibility of node i as

$$\begin{aligned} \textit{Flexibility}[i] = \alpha \times \textit{Range}_{\mathrm{A}}[i] \\ + \beta \times \textit{Range}_{\mathrm{E}}[i] + \gamma \times \textit{Range}_{\mathrm{D}}[i] \end{aligned}$$

where α, β, and γ are weights given by users and $0 \leq \alpha$, β, and $\gamma \leq 1$. A node i is called a flexible node if it satisfies $\textit{Flexibility}[i] \geq \lambda$, where λ is a threshold specified by users. Furthermore, we define that a path is a *flexible path* if all the nodes on the path are flexible nodes. The advantage of travelling along a flexible path is that no matter where we are in the path we have flexibility to change our schedule to cope with future unexpected delay or fluctuation in travel time. And we call a flexible path from nodes 1 to i as the most flexible path if it has the greatest flexibility at node i.

Next, we study the problem of finding the most flexible path from nodes 1 to i. Note that the three time ranges are computed from the six time characteristics LA_i, EA_i, LE_i, EE_i, LD_i, and ED_i. Furthermore, these six characteristics are computed from Algorithms I–III, where Algorithm I computes EE_i and ED_i from EA_i, Algorithm II computes LE_i and LD_i, and Algorithm III derives LA_i from LE_i and LD_i. From these three algorithms, we know that LA_i, LE_i, and LD_i are not dependent on EA_i; but EE_i and ED_i are positively related to EA_i. In other words, if EA_i becomes smaller, then EE_i and ED_i will become smaller or remain the same, but LA_i, LE_i and LD_i will not change. As a result, the smaller EA_i is, the wider the three intervals are, or the more flexible node i is. From this simple but important observation, we conclude that the flexible path that leads to the earliest arrival at node i is the most flexible path to node i. Therefore, our problem now becomes how to find the flexible path with the earliest arrival time at node n. For ease of reference, we call the path the *most flexible path* to node n.

The most flexible path to a node, say node j, denoted as $MFP(1, j)$, has an attractive property that any prefix path of $MFP(1, j)$ to an intermediate node i can be replaced by the most flexible path to node i, that is, $MFP(1, i)$, without increasing the earliest time to arrive node j. By going through $MFP(1, i)$ other than the original prefix path, we will arrive node i earlier. Then, we can have the same earliest arrival time to node j just by waiting outside node i until the original arrival time has reached and then tracing the remaining path in the original way.

The above observation derives a very useful property that a prefix path of the most flexible path is also the most flexible path. Consequently, the proposed algorithm first finds the most flexible path for shorter paths and, based on the result, gradually extends to longer paths. The algorithm is described as follows. Note that we assume that LA_i, LE_i, and LA_i have been calculated by Algorithms II and III beforehand.

Algorithm Finding-path

1. Set $EA_1 = 0$.
   Set all $EA_u = \infty$ for all nodes u in N.
   Insert all values of EA_u into the set HP.
2. Find and remove the minimum element EA_u from HP.
3. Determine whether node u is flexible by the following steps.
   3.1. Compute EE_u and ED_u using EA_u.
   3.2. Retrieve the values of LA_i, LE_i, and LD_i.
   3.3. Compute the three time inter vals and node's flexibility.
4. If $u = n$ then go to step 6.
5. If node u is not flexible then go to step 2; else do the following.
   For each arc (u, w) emanating from node u, do
   $\text{temp}_w = \text{ED}_u + \text{dur}(u, w)$.
   If $temp_w < EA_w$ then
   $\text{EA}_w = \text{temp}_w$, $\text{pred}_w = \text{u}$, and
   update the value of EA_w in HP.
   Go to step 2.
6. If node n is flexible then output the most flexible path by tracing back through $pred_n$;
   Else the network has no flexible path to node n.

Example 2

Consider the network shown in Figure 4, where the arc's time is shown along each arc, the attached windows are indicated next to each node, and the specified deadline is 25. Beside each node, we also show its latest entering time and latest departure time, which are calculated by Algorithm II. Further, we assume $\alpha = 0$, $\beta = 1$, $\gamma = 1$, and $\lambda = 4$ in determining a node's flexibility. Under this assumption, the flexibility of a node is computed by summing the ranges of the departure and entering time intervals, and if the total value is smaller than 4, it is not flexible. The execution of the algorithm is shown in Table 3, where there are seven iterations, and, in each iteration, the node with the minimal EA value is removed from HP. Moreover, in each iteration, we also check whether the chosen node is flexible. If a node is not flexible, we skip it and go to the next iteration, because it cannot be an intermediate node of a flexible path. Otherwise, we will update the earliest arrival times of its adjacent nodes, if it can provide a shorter path by going through this chosen node. Following this procedure, we found that path (1, 3, 5, 7) is the most flexible path. It is interesting to note that there are actually two paths that arrive at node 7 earlier than this path, that is, path (1, 3, 5, 6, 7) and path (1, 2, 5, 7). These two paths were not selected, however, because the former has an inflexible node 6 and the latter has an inflexible node 2 as their intermediate nodes.

The following theorem shows the validity of the algorithm.

Theorem 1 *Algorithm Finding-path finds the most flexible path in a network.*

Proof. We prove the result by induction. At any given iteration, the algorithm partitions all nodes into two sets, namely, HP and $\overline{HP}$, where $\overline{HP} = N - HP$. Our induction hypotheses are founded on the premises that: (1) the time

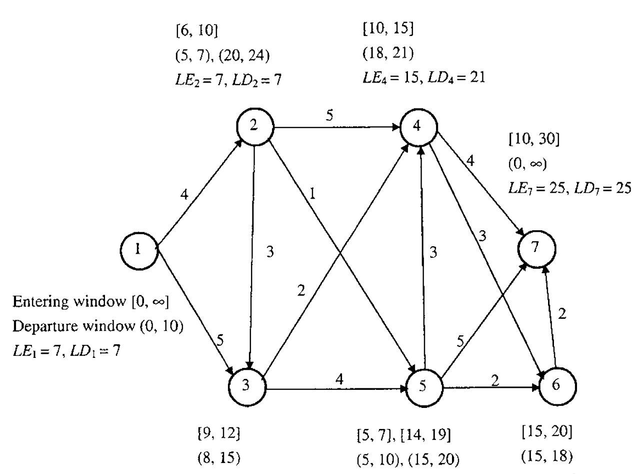

Figure 4 Network after preprocessing.

Table 3 Execution of Algorithm Finding-path

Iteration	*Nodes*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*Chosen node*	*Flexibility of the node*	*Flexible or not*
	EA	0	4	5	∞	∞	∞	∞			
	EE	0	6	9	∞	∞	∞	∞			
1	ED	0	6	9	∞	∞	∞	∞	1	$14=7+7$	Yes
	Belong to	$\overline{HP}$	HP	HP	HP	HP	HP	HP			
	EA	0	4	5	∞	∞	∞	∞			
	EE	0	6	9	∞	∞	∞	∞			
2	ED	0	6	9	∞	∞	∞	∞	2	$2=1+1$	Not
	Belong to	$\overline{HP}$	$\overline{HP}$	HP	HP	HP	HP	HP			
	EA	0	4	5	11	13	∞	∞			
	EE	0	6	9	11	14	∞	∞			
3	ED	0	6	9	18	15	∞	∞	3	$9=3+6$	Yes
	Belong to	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	HP	HP	HP	HP			
	EA	0	4	5	11	13	21	22			
	EE	0	6	9	11	14	∞	22			
4	ED	0	6	9	18	15	∞	22	4	$7=4+3$	Yes
	Belong to	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	HP	HP	HP			
	EA	0	4	5	11	13	17	20			
	EE	0	6	9	11	14	17	20			
5	ED	0	6	9	18	15	17	20	5	$10=5+5$	Yes
	Belong to	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	HP	HP			
	EA	0	4	5	11	13	17	20			
	EE	0	6	9	11	14	17	20			
6	ED	0	6	9	18	15	17	20	6	$2=1+1$	Not
	Belong to	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	HP			
	EA	0	4	5	11	13	17	20			
	EE	0	6	9	11	14	17	20			
7	ED	0	6	9	18	15	17	20	7	$10=5+5$	Yes
	Belong to	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$	$\overline{HP}$			

label EA_u of each node u in $\overline{HP}$ is the earliest possible time to arrive node u through a flexible path, and (2) the time label EA_u of each node u in $\overline{HP}$ is the earliest time to arrive node u through a flexible path, provided that each intermediate node in the path lies in $\overline{HP}$. We perform induction according to the cardinality of the set $\overline{HP}$.

To prove hypothesis (1), recall that, in each iteration, we move a node u in HP with the smallest value to $\overline{HP}$. To show that EA_u of node u is optimum, notice that, by our induction hypothesis (2), EA_u is the earliest time to arrive node u through a flexible path that does not contain any intermediate node in HP. We now show that the total time to arrive node u through any flexible paths that contain some nodes in HP as an intermediate node is at least EA_u. To prove it, we let P denote a path formed by appending node u after a flexible path, and assume that P contains at least one node in HP as an intermediate node. Path P can be decomposed into two segments P_1 and P_2, where all intermediate nodes in P_1 are not in HP, but the last node of P_1, say h, is in HP. By the induction hypotheses, this suggests that the total time of P_1 is at least EA_h. Moreover, since node u is the smallest time label in HP, $EA_h \geq EA_u$. Therefore, the path segment P_1 has total time of at least EA_u. Furthermore, since all arc times are nonnegative, the total time of the path segment P_2 is nonnegative. Consequently, the total time of path P is no less than EA_u. This result establishes the fact that EA_u is the earliest time to arrive node u through a flexible path.

We next show that the algorithm preserves hypothesis (2). In step 2, the node with smallest EA_u is selected, and its value becomes permanent thereafter. Following that, we will check if this node is flexible. If it is not flexible, then this node cannot be used as an intermediate node of a flexible path, and thus we skip it and move on to remaining nodes. Otherwise, the time labels of some nodes in $HP-\{u\}$ may decrease since node u could become an intermediate node in the most flexible paths to these nodes. Recall that after permanently labeling flexible node u, the algorithm examines each arc (u, w) emanating from node u and sets $EA_w = ED_u + dur(u, w)$ if $ED_u + dur(u, w) < EA_w$. Therefore, after this time-label update operation, the time label of each node w in $HP-\{u\}$ is the earliest possible time to arrive node w among all flexible paths whose intermediate nodes are all in $\overline{HP} \cup \{u\}$. □

Finally, the following lemma shows the time complexity of the algorithm.

Lemma 1 *Let n denote the number of nodes and m the maximum number of time windows associated with a node in the network. If* $n \geq m$*, then the time complexity of Algorithm Finding-path is* $O(n^2)$*.*

Proof. The algorithm requires n iterations to find the most flexible path, since the initial HP has n nodes, and one node is removed from HP in each iteration. There are three operations in each iteration: (1) select the node with the minimum EA, (2) compute the flexibility of node u, and (3) update EA values of those nodes adjacent to node u. Obviously, (1) can be done in time O(n), because finding the minimum value from n values can be done in time O(n), and (3) can also be done in time O(n) because each node has no more than n adjacent nodes. As to (2), it can be done in time O(m). To explain the time complexity associated with (2), note that the most time-consuming part for (2) is in step 3.1, in which EE_u and ED_u are derived from EA_u. To perform this computation, all entering time windows and departure time windows of node u need to be examined once. As a result, the time required is O(m). Combining all the above results, we have the total time complexity of O(n^2).

Multiple intermediate destination nodes

In this subsection, we discuss the method of selecting a path when multiple intermediate destination nodes with multiple deadlines are pre-specified. Under this new requirement, it is necessary to evaluate the changes to the time characteristics. Let the specified intermediate nodes be I_j with the latest entering time E_j, for $j = 1$ to r. Without loss of generality, let $I_0 = 1$ and $I_r = n$. Consequently, the path must be from node I_0 to node I_1, then to $I_2, \ldots$, and finally to I_r. In this path, we may pass through a node several times. Therefore, a node should have r sets of time characteristics rather than just only one set, where the jth set is for the subpath from I_{j-1}. to I_j. In view of this, we define new symbols, $EA_i(j)$, $EE_i(j)$, $ED_i(j)$, $LD_i(j)$, $LE_i(j)$, and $LA_i(j)$, which are similar to those defined previously except that they are constrained in their respective subpaths.

To compute $EA_i(j)$, $EE_i(j)$, and $ED_i\{j\}$ for $j = 1, 2, \ldots$, and r, we execute Algorithm I r times, where, in the jth run, the source node s is I_{j-1}, destination node d is I_j, and the beginning time of source node is $ED_s(j-1)$. In order to compute $LE_i(j)$ and $LD_i(j)$, we execute Algorithm II backward for $j = r, r-1, \ldots, 1$. In the run corresponding to index j, we set the source node s as I_{j-1}, destination node d as I_j, and the latest entering time of destination node as $\min\{LE_d(j+1), E_j\}$. Finally, by applying Algorithm III r times, we can obtain all the values for $LA_i(j), j = 1, 2, \ldots, r$.

Based on the time characteristics $LD_i(j)$, $LE_i(j)$, and $LA_i(j)$, where $j = 1, 2, \ldots, r$, Algorithm Finding-path can find the most flexible path for each segment, that is, the most flexible paths from I_{j-1} to I_j, for $j = 1, 2, \ldots, r$. In the jth run, the source node is I_{j-1}, destination node is I_j, and the beginning time of the source node is the earliest departure time of the node in the preceding run. After all these sub-paths are joined one after another, the final path is the one that is from node I_0 to node I_1, then to $I_2, \ldots$, and finally to I_r.

Example 3

Consider the network shown in Figure 1. Suppose we want to find a path with two destinations, where the first destination is $I_1 = 6$ with $E_1 = 25$ and the second $I_2 = 12$ with $E_2 = 40$. Running Algorithm I directly for these two segments, we find that the optimal paths in the segments are (1, 2, 3, 6) and (6, 5, 12), respectively.

Following the method discussed in this subsection, we run Algorithm II backward two times. In the first time, we set the source node as node 6, the destination node as node 12, and $LE_{12}(2) = 40$. As a result, the latest entering times and the latest departure times for nodes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12 are found to be (10, 10), (15, 16), (13, 18), (16, 18), (28, 28), (23, 24), (22, 22), (18, 18), (15, 25), (25, 28), (22, 30), and (40, 40), respectively, where the first number in the parentheses is the latest entering time and the second the latest departure time. In running Algorithm II for the second time, we set the source node as node 1, the destination node as node 6, and $LE_6(1) = \min\{LE_6(2), E_2\} = \min\ \{23, 25\} = 23$. We find that the latest entering times and the latest departure times for nodes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12 are (4, 4), (5, 5), (11,11), $(-\infty, -\infty)$, $(-\infty,-\infty)$, (23, 23), (10, 10), $(-\infty,-\infty)$, $(-\infty,-\infty)$, $(-\infty,-\infty)$, $(-\infty,-\infty)$, and $(-\infty,-\infty)$, respectively.

Suppose a node's flexibility is determined by the following parameter values: $\alpha = 0$, $\beta = 1$, $\gamma = 1$, and $\lambda = 4$. Using these values, the flexibility of a node is calculated by summing the ranges of the departure and entering time intervals, and a node is not flexible if the total is smaller than 4. Algorithm Finding-path needs to be run twice, where the first run is used to find the most flexible path from nodes 1 to 6, and the second is used for that from nodes 6 to 12. In the first run, we find that the most flexible path is path (1, 3, 6) rather than the original optimal path (1, 2, 3, 6). This is because node 2 is not flexible. (Node 2 has the following time characteristics: $EE_1(2) = 3$, $ED_1(2) = 4$, $LE_1(2) = 5$ and $LD_1(2) = 5$.) In the second run, we set $ED_2(6)$ as 20, because the result of the preceding run indicates that 20 is the earliest departure time of node 6. The result shows that the most flexible path from nodes 6 to 12 is path (6, 5, 12). Combining the two paths together, we obtain the final path (1, 3, 6, 5, 12), where every node in the path is flexible and the latest entering time constraints on nodes 6 and 12 are satisfied.

Conclusion

In this paper, we propose a systematic method for planning a path in a time-window network to meet a pre-determined deadline constraint. The main results of the paper include (1) three efficient algorithms developed for generating six basic time characteristics associated with each node, (2) a systematic analysis proposed to generate important time information for planning a path in a time-window network, and (3) based on the time characteristics and the proposed analyses, an algorithm to find the most flexible path in a time-window network.

The paper can be extended in several ways. For example, we may examine other kinds of time constraints, such as time-schedule and traffic-light constraints, and see how this time-related analysis can be done in these time-constrained networks. Furthermore, we may develop systematic or interactive procedures for selecting a path according to these time characteristics. In addition, we may generalize the problem to multiple travelers by requiring the travelers must meet at certain nodes before specified deadlines.

References

1 Baker EK (1983). An exact algorithm for the time-constrained traveling salesman problem. *Opns Res* **31**: 938–945.
2 Baker E (1982). Vehicle routing with time window constraints. *Logist Transb Rev* **18**: 385–401.
3 Ahn BH and Shin JY (1991). Vehicle-routing with time windows and time-varying congestion. *J Opl Res Soc* **42**: 393–400.
4 Balakrishnan N (1993). Simple heuristics for the vehicle routing problem with soft time windows. *J Opl Res Soc* **44**: 279–287.
5 Desrochers M and Soumis F (1988). A reoptimization algorithm for the shortest path problem with time windows. *Eur J Opl Res* **35**: 242–254.
6 Dumas Y, Desrosiers J, Gelinas E and Solomon MM (1995). An optimal algorithm for the traveling salesman problem with time windows. *Opns Res* **43**: 367–371.
7 Desaulniters G and Villeneuve D (2000). The shortest path problem with time windows and linear waiting costs. *Transp Sci.* **34**: 312–319.
8 Dijkstra EW (1959). A note on two problems in connection with graphs. *Numer Math* **1**: 269–271.
9 Fox BL (1978). Data structures and computer science techniques in operations research. *Opns Res* **26**: 686–717.
10 Eppstein D (1994). Finding the k shortest paths. *Proceeding of the 35th Annual Symposium on Foundations of Computer Science*. pp 154–165.
11 Chen YL Rinks D and Tang K (1997). Critical path in an activity network with time constraints. *Eur J Opl Res* **100**: 122–133.
12 Chen YL and Tang K (1998). Minimal time paths in a network with mixed time constraints. *Comput Opns Res* **25**: 793–805.
13 Chen YL and Yang HH (2000). Shortest paths in traffic-light networks. *Transp Res:B* **34**: 241–253.

Appendix

Algorithm I. Evaluation of EA_i, EE_i, and ED_i:

Step 1: Initialization
  For each node i, except the source node, set $EA_i=\infty$, $EE_i=\infty$, and $ED_i=\infty$;
  $EA_1=0$; $EE_1=0$; $ED_1=0$;
  Mark all the nodes as unexamined nodes;

Step 2: Let node i be the node with the minimum ED_i value among the unexamined nodes;
  For each unexamined successor j of node i, do
    if $ED_i+dur(i,j)<EA_j$, then
      $EA_j=ED_i+dur(i,j)$;
      If $EA_j>Eend_j^{E_j}$, then go to the next iteration of the for-loop;
      Find the minimal $k(1\le k\le E_j)$, such that $EA_j\le Eend_j^k$;
      $EE_j=\max\{EA_j, Ebegin_j^k\}$;
      If $EE_j>Dend_j^{E_j}$, then go to the next iteration of the for-loop;
      Find the minimal $k(1\le k\le D_j)$, such that $EE_j\le Dend_j^k$;
      $ED_j=\max\{EE_j, Dbegin_j^k\}$;
      If ED_j has been modified, then $EP_j=i$;
      /* EP_j is used to store the earliest path route */
    Mark node i as an examined node;

Step 3: Repeat step 2 until all the nodes have been examined;

Algorithm II. Evaluation of LE_i and LD_i:

Step 1: Initialization
  For each node i other than destination n, do $LE_i=-\infty$ and $LD_i=-\infty$;
  $LE_n=Eend_j^{E_n}$;
  Mark all the nodes as unexamined nodes;

Step 2: Backward computation
  Let node i be the node with the maximum LE_i among the unexamined nodes;
  For each unexamined predecessor j of i, do
    if $LE_i–dur(j,i)>LD_j$, then
      If $LE_i–dur(j,i)<Dbegin_j^1$, then go to the next iteration of the for-loop;
      Find the maximal $k(1\le k\le D_j)$ such that $Dbegin_j^k\le LE_i–dur(j,i)$;
      $LD_j=\max\{LD_j, \min\{LE_i–dur(j,i), Dend_j^k\}\}$;
      If $LD_j<Dbegin_j^1$, then go to the next iteration of the for-loop;
      Find the maximal $k(1\le k\le E_j)$, such that $Ebegin_j^k\le LD_j$;
      $LE_j=\max\{LE_j, \min\{LD_j, Eend_j^k\}\}$;
      If LE_j is changed, then set $LP_j=i$;
      /* LP_j stores the latest path route */
  Mark node i as an examined node;

Step 3: Repeat step 2 until all the nodes have been examined.

Algorithm III. Evaluation of LA_i:

For each node i except the source node, do
begin
  $LA_i=-\infty$;
  For each predecessor node j of i, do
    $x=LE_i–dur(j,i)$;
    if $x\ge LD_j$ then $y=LD_j$
    else if $x\ge ED_j$ then begin

Find the maximal k such that $Dbegin_j^k \leq x$;
$y = \min\{Dend_j^k, x\}$
end
else $y = -\infty$;
$LA_i = \max\{LA_i, y + dur(j, i)\}$;
if LA_i, is changed, then set $LAP_i = j$ /* LAP_i is used to store the node from which we
can arrive at node i in the latest time. */
end

Acknowledgments—We thank two anonymous referees for their many helpful suggestions that improved this paper. The first author was supported, in part, by the Ministry of Education (MOE) Program for Promoting Academic Excellence of Universities under the Grant Number 91-H-FA07-1-4.

Received January 2001
accepted February 2003 After three revisions