Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computers & Operations Research 31 (2004) 2405–2418

computers & operations research

www.elsevier.com/locate/dsw

A label-setting algorithm for finding a quickest path

Chan-Kyoo Park[a], Sangwook Lee[b,*], Soondal Park[b]

[a]*Department of IT Audit and Supervision, National Computerization Agency, NCA Building, 77, Mugyo-dong, Jung-gu, Seoul 100-170, South Korea*

[b]*Department of Industrial Engineering, Seoul National University, Kwanak-gu, Seoul 151-742, South Korea*

Abstract

The quickest path problem is to find a path to send a given amount of data from the source to the destination with minimum transmission time. To find the quickest path, existing algorithms enumerate non-dominated paths with distinct capacity, and then determine a quickest path by comparing their transmission time. In this paper, we propose a label-setting algorithm for finding a quickest path by transforming a network to another network where an important property holds that each subpath of a quickest path is also a quickest path. The proposed algorithm avoids enumerating non-dominated paths whose transmission time is greater than the minimum transmission time. Although the computational complexity of the proposed algorithm is the same as that of existing algorithms, experimental results show that our algorithm is efficient when a network has two or more non-dominated paths.

Keywords: Quickest path; Optimal path; Communication network; Data transmission; Label-setting algorithm

1. Introduction

The quickest path problem (QPP) is to find a path to send a given amount of data from the source to the destination with minimum transmission time in a network where each arc has a lead time and capacity. The transmission time of a path depends on both its lead time and capacity. In that respect, the quickest path can be regarded as a bicriterion path problem of which objectives are to minimize the length of paths and to maximize the capacity of paths between the origin and the destination.

Let $G=(N,E)$ be a directed network where N and E denote the set of nodes and arcs, respectively. For each arc $(i,j)\in A$, nodes i and j are called the *head* and *tail* of the arc, respectively. We assume

* Corresponding author.
E-mail addresses: parkck@nca.or.kr (C.-K. Park), lessw@orlab.snu.ac.kr (S. Lee), sdpark@orlab.snu.ac.kr (S. Park).

0305-0548/$ - see front matter © 2003 Published by Elsevier Ltd.
doi:10.1016/S0305-0548(03)00195-3

without loss of generality that G has no parallel arcs, that is, no two arcs have the same tail and head nodes because any directed network with parallel arcs can be easily transformed into another network without parallel arcs. For an arc (i,j), $l(i,j) \geqslant 0$ and $c(i,j) > 0$ denote the lead time and capacity of (i,j). A sequence of distinct nodes, $p = (u = u_1, u_2, \ldots, v = u_k)$ with $k \geqslant 2$, is called a *path* from u to v or a *u–v path* if it satisfies the condition that $(u_i, u_{i+1}) \in A$ for each $1 \leqslant i \leqslant k-1$. The lead time of path p is defined as

$$l(p) = \sum_{i=1}^{k-1} l(u_i, u_{i+1})$$

and the capacity of path p is defined as

$$c(p) = \min_{1 \leqslant i \leqslant k-1} c(u_i, u_{i+1}). \tag{1}$$

Then, the transmission time required to send σ units of data through path p is defined as

$$t(p) = l(p) + \frac{\sigma}{c(p)}.$$

Although QPP has wide range of applications for routing in communication and transportation networks, it had not received much attention until Chen and Chin [1] suggested their algorithm. In fact, QPP was originally suggested by Moore [2], who also proposed an algorithm whose time complexity is actually $O(hm + hn \log n)$ where $|N| = n$, $|E| = m$, and h is a parameter that never exceeds the number of distinct capacities greater than the capacity of the shortest path, with respect to lead time, in the original network. Chen and Chin [1] proposed an algorithm with time complexity $O(rm + rn \log m)$ and additional space requirement of $O(rm)$ for a given value of σ where $r(\leqslant m)$ denotes the number of distinct capacity values in the network G. In Chen and Chin's algorithm, the original network is transformed into another network by creating r copies for each node and arc in G. In the transformed network, we consider only the lead time of each arc because the capacity of any path can be automatically calculated from the nodes which it passes through. Rosen et al. [3] developed an alternative algorithm for QPP with time complexity $O(rm + rn \log m)$ and additional space requirement of $O(n)$. Martins and Santos [4] regarded QPP as a bicriterion path problem, and proposed an algorithm which is similar to Rosen et al.'s algorithm. As further studies on extended problems of QPP, Chen and Hung [5] presented an $O(mn^2)$ time algorithm that finds all-pairs quickest paths, and Chen [6] presented an $O(kmn^3 + kmn \log k)$ algorithm that finds the first k quickest simple paths for a given pair of nodes. Kagaris et al. [7] considered the minimum transmission time problem, a generalization of QPP where data can be transmitted through more than one path, and showed that the problem is NP-complete. Xue et al. [8] showed that the minimum transmission time problem with integer capacities and integer delays is equivalent to the maximal dynamic flow problem. In addition, Lin [9] proposed a variation of QPP where the capacity of each arc is stochastic, not deterministic.

In this paper, we are concerned with two drawbacks of existing algorithms. First, to find a quickest path, the existing algorithms enumerate non-dominated paths with distinct capacity where a path p is said to be *non-dominated* if there exists no path p' such that $l(p') < l(p)$ and $c(p') \geqslant c(p)$, or $l(p') = l(p)$ and $c(p') > c(p)$. One of the non-dominated paths is selected as a quickest path by comparing their transmission time required to send a given σ units of data. This approach seems to be efficient for finding quickest paths for all values of σ, but not for the case where σ is fixed because unnecessary non-dominated paths, which are not quickest, have to be enumerated. On the other hand, Chen and Chin [1] showed that a Dijkstra-like labeling algorithm might fail to solve QPP because QPP does not satisfy the property that the subpath of a quickest path is also a quickest path.

In this paper, we propose a label-setting algorithm, like Dijkstra's algorithm, that can solve QPP without enumerating non-dominated paths. The main idea of the proposed algorithm is that a simple modification of Chen and Chin [1] method leads to another transformed network which has the nice property that the subpath of a quickest path is also a quickest path. By applying the label-setting algorithm to QPP in the transformed network, the enumeration of unnecessary non-dominated paths can be avoided. In addition, to reduce the amount of storage space required, we propose an efficient implementation algorithm which can solve QPP by performing the transformation of the original network implicitly.

The organization of this paper is as follows: In Section 2, we propose a simple label-setting algorithm for QPP which can be applied to transformed networks. In Section 3, we present an efficient implementation of the label-setting algorithm by performing the transformation of the original network implicitly and provide an example which illustrates how the proposed algorithm works. In Section 4, we present experimental results that show the efficiency of the proposed algorithm. Finally, some concluding remarks are given in Section 5.

2. A simple label-setting algorithm for QPP

We begin with an example in Fig. 1 that illustrates how existing algorithms find a quickest path. The source and destination are node 1 and 7, respectively.

In Fig. 1, there are four non-dominated paths from node 1 to node 7 as shown in Table 1. Path $1 \to 3 \to 6 \to 7$ is the quickest one, but its subpath, $1 \to 3 \to 6$, is not the quickest path from node 1 to node 6, which makes a Dijkstra-like labeling algorithm fail to solve QPP. On the other hand, Chen and Chin's algorithm (CCA) [1] enumerates four non-dominated paths presented in Table 1, although it searches them in the transformed network instead of the network in Fig. 1. Rosen et al.'s algorithm [3] and Martins and Santos' algorithm (MSA) [4] enumerate the first three non-dominated paths except the last path in Table 1.

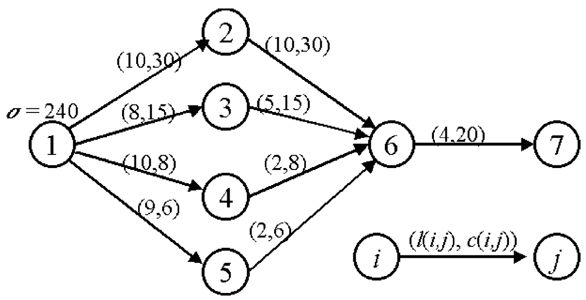

Fig. 1. An example for the quickest path problem.

Table 1
The non-dominated paths found by existing algorithms

Non-dominated path	Lead time	Capacity	Transmission time
$1 \to 5 \to 6 \to 7$	15	6	55
$1 \to 4 \to 6 \to 7$	16	8	46
$1 \to 3 \to 6 \to 7$	17	15	33
$1 \to 2 \to 6 \to 7$	24	20	36

```
Step 1. Expand nodes and arcs.
  $N' = \{x^{c_1}, x^{c_2}, \dots, x^{c_r} | x \in N\}$.
  $E' = \{(x^{c_i}, y^{c_i}), (x^{c_j}, y^{c_k}) | (x,y) \in E, 1 \le i \le k$ and $k+1 \le j \le r$
      where $k$ is an integer such that $c_k = c(x,y)\}$.
  $l(x^{c_i}, y^{c_j}) = l(x,y)$ for all $(x^{c_i}, y^{c_j}) \in E'$.
  $c(x^{c_i}, y^{c_j}) = \min\{c_i, c_j\}$ for all $(x^{c_i}, y^{c_j}) \in E'$.

Step 2. Add an artificial node $(n+1)$ and its incident arcs $(t^{c_i}, n+1)$ for
      each node $t^{c_i}$.
  $N' = N' \cup \{n+1\}$.
  $E' = E' \cup \{(t^{c_i}, n+1) | i = 1, \dots, r\}$.
  $l(t^{c_i}, n+1) = 0, i = 1, \dots, r$.
  $c(t^{c_i}, n+1) = c_i, i = 1, \dots, r$.
```

Fig. 2. Transformation procedure.

The existing algorithms have a common point that they neither calculate the transmission time of paths nor make use of this information until all non-dominated paths are found. By using the transmission time of paths, however, it is possible to avoid enumerating unnecessary non-dominated paths that have the transmission time greater than the minimum transmission time. For example, the transmission time of path $1 \to 4$ is 40, which is greater than the minimum transmission time, 33, of the quickest path. Also, the transmission time of path $1 \to 5$ is 49. The non-nominated paths which include the subpaths, such as $1 \to 4$ and $1 \to 5$, cannot be a quickest one, and consequently need not be enumerated.

We already know that a direct application of label-setting approach to QPP might fail to find a quickest path. However, it can be successfully applied to QPP in a transformed network. We explain how to construct the transformed network from a given network. Let $\{c_1, c_2, \dots, c_r\}$ be the set of distinct capacity values in G. We assume without loss of generality that $c_1 < c_2 < \cdots < c_r$. The procedure for constructing the transformed network $G' = (N', E')$ from $G = (N, E)$ is presented in Fig. 2.

The origin and destination in the transformed network are s^{c_r} and $(n+1)$, respectively. Note that Step 1 in Fig. 2 is similar to Chen and Chin's transformation procedure [1], but Step 1 sets each arc's capacity explicitly. For example, consider a network that is identical to the network presented in Fig. 1 except that node 5 and its incident arcs are removed for the sake of simplicity. Then, it is transformed into the network in Fig. 3 by the transformation procedure.

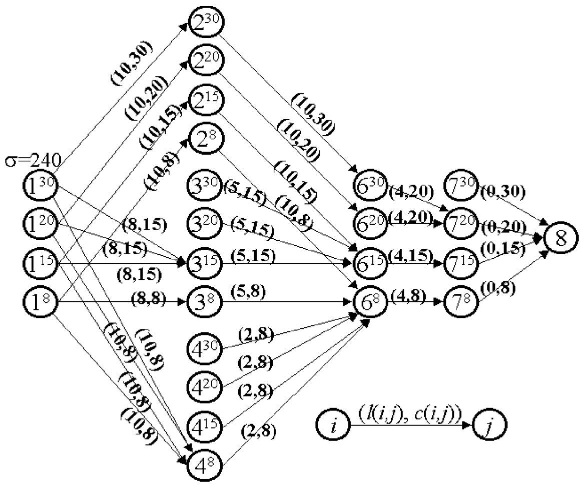

Fig. 3. An example of the transformation procedure.

Recall that for all $(x^{c_i}, y^{c_j}) \in E'$, $c(x^{c_i}, y^{c_j})$ is set to $\min\{c_i, c_j\}$ by the transformation procedure in Fig. 2. This rule, together with the definition of E', makes the transformed network have the property that the capacity of any path in the transformed network is determined by its last arc's capacity. In addition, the transformed network has another important property that "*any subpath of a quickest path is also a quickest path*", which is shown in Theorem 1.

Theorem 1. *Let p be a quickest path of G'. Each subpath of p is also a quickest path of G'.*

Proof.

(i) In case that the last node of p is not node $(n+1)$.
Let $p=(x_0^{c_{i_0}}, x_1^{c_{i_1}}, \dots, x_k^{c_{i_k}})$. Let q be an arbitrary subpath of p such that $q=(x_u^{c_{i_u}}, x_{u+1}^{c_{i_{u+1}}}, \dots, x_v^{c_{i_v}})$ and $0 \leqslant u < v \leqslant k$. Suppose that q is not a quickest path. Let q' be a quickest path from $x_u^{c_{i_u}}$ to $x_v^{c_{i_v}}$ in G'. Since the destinations of q and q' are the same, $c(q)=c(q')=c_{i_v}$. Hence, $l(q) > l(q')$. After replacing the subpath q of p by q', we can obtain another path p' such that $l(p) > l(p')$ and $c(p)=c(p')$, which implies that $t(p) > t(p')$. This is contradictory to the assumption. Therefore, each subpath of p is also a quickest path of G'.

(ii) In case that the last node of p is node $(n+1)$.
Let $p=(x_0^{c_{i_0}}, x_1^{c_{i_1}}, \dots, x_k^{c_{i_k}}, n+1)$. For any subpath q which does not pass through the last arc $(x_k^{c_{i_k}}, n+1)$, we can show by a similar argument as (i) that q is also a quickest path. Let q be a subpath of p such that $q=(x_u^{c_{i_u}}, x_{u+1}^{c_{i_{u+1}}}, \dots, x_k^{c_{i_k}}, n+1)$ and $0 \leqslant u \leqslant k$. Suppose q is not a quickest path. Let q' be a quickest path from $x_u^{c_{i_u}}$ to $(n+1)$ in G'.

(a) If q' goes through node $x_k^{c_{i_k}}$, then $c(q)=c(q')$ and $l(q) > l(q')$. By the same argument as used in (i), we can obtain another path p' such that $l(p) > l(p')$, $c(p)=c(p')$, and $t(p) > t(p')$. This is contradictory to the assumption. Therefore, q is also a quickest path of G'.

```
Algorithm. SLSA(Simple Label-Setting Algorithm)
1  $S \leftarrow \emptyset, \bar{S} \leftarrow N'$
2  $d(x) \leftarrow \infty$ for each node $x \in N'$
3  $d(s^{c_r}) \leftarrow 0$ and $pred(s^{c_r}) \leftarrow 0$
4  while $(n+1) \notin S$ do begin
5    let $x^* \in \bar{S}$ be a node such that $d(x^*) = \min\{d(x) | x \in \bar{S}\}$
6    $S \leftarrow S \cup \{x^*\}$
7    $\bar{S} \leftarrow \bar{S} - \{x^*\}$
8    for each $(x^*, y) \in A_{G'}(x^*)$ do
9      if $d(y) > (d(x^*) - \frac{\sigma}{c(pred(x^*), x^*)}) + l(x^*, y) + \frac{\sigma}{c(x^*, y)}$, then
10       $d(y) \leftarrow (d(x^*) - \frac{\sigma}{c(pred(x^*), x^*)}) + l(x^*, y) + \frac{\sigma}{c(x^*, y)}$
11       $pred(y) \leftarrow x^*$
12     end if
13   end for
14 end while
```

Fig. 4. A simple label-setting algorithm for QPP.

(b) If q' doesn't go through node $x_k^{c_{i_k}}$, it is possible that the capacity of q' does not equal that of q. By the assumption, we have

$$t(q) = l(q) + \frac{\sigma}{c(q)} > l(q') + \frac{\sigma}{c(q')} = t(q'). \tag{2}$$

Let p' be a path which is obtained by replacing the subpath q of p by q'. Recall that $c(p) = c(q)$ and $c(p') = c(q')$, since q and q' go through the last arcs of p and p', respectively. Adding $l(p) - l(q)$ to both sides of inequality (2), we obtain

$$(l(p) - l(q)) + l(q) + \frac{\sigma}{c(q)} > (l(p) - l(q)) + l(q') + \frac{\sigma}{c(q')}$$

$$\Leftrightarrow t(p) > t(p'),$$

which contradicts the assumption that p is a quickest path. Therefore, q is also a quickest path. □

Using Theorem 1, we can develop a Dijkstra-like label-setting algorithm for finding a quickest path in the transformed network as presented in Fig. 4. Let $A_{G'}(x)$ denote the set of arcs emanating from x in G', i.e., $A_{G'}(x) = \{(x, y) | (x, y) \in E'\}$. We define $c(0, x) = \infty$ for an arbitrary node x, and σ/∞ is defined to be 0. A label $d(x)$ represents the transmission time required to send data from the source to x through the path p which is the quickest among the paths found until each iteration. Path p can be constructed recursively by $pred(x)$.

We can show the correctness of SLSA by the same argument that is used by Hu [10] for the proof of the correctness of Dijkstra algorithm. Hu's argument is based only on the property that each subpath of a shortest path must be also a shortest path. Hence, the correctness of SLSA can be directly shown using Theorem 1 and Hu's argument.

The transformed network has $(rn + 1)$ nodes and $(rm + r)$ arcs. Since the running time of the best polynomial time algorithm for solving the shortest path problem in $G = (N, E)$ is known to be $O(m + n \log n)$ [11], the computational complexity of SLSA is $O(rm + rn \log rn)$, which is the same as that of Chen and Chin's algorithm (CCA). However, SLSA has two advantages in respect of the average performance: First, it can find a quickest path without enumerating non-dominated paths. While CCA has to find a shortest path from s^{c_r} to t^{c_i} for each $i = 1, \dots, r$, SLSA finds only one non-dominated path which is also a quickest path. This situation is similar to the case of the shortest path problem where finding a shortest path from the source to one destination is more efficiently solved than finding shortest paths from the source to the other nodes, but both the problems have the same computational complexity. The other advantage is that SLSA calculates the transmission time of each path from the source to other nodes, and makes use of this information to prevent paths with too great transmission time from being searched. This feature will reduce the running time of SLSA significantly.

3. An efficient implementation of the simple label-setting algorithm for QPP

Although SLSA has two advantages in respect of performance, it has two problems to be resolved for its efficient implementation. First, SLSA can be applied only after the original network is transformed using the procedure given in Fig. 2. The transformation may seem some tedious work. The other problem is that the amount of storage space for SLSA is $O(rm + rn)$. However, as shown in Fig. 3, some nodes and arcs in the transformed network need not be generated. For example, nodes $3^{30}, 3^{20}, 4^{30}, 4^{20}$ and their incident arcs can be removed without impairing the correctness of SLSA.

The problems above can be resolved by developing an efficient implementation of SLSA which runs without transforming network explicitly. The basic idea is that each node is allowed to have multiple labels and each label is created only when a path to the node is found. Each label of a node keeps the transmission time required to send data from the source to the node through a certain path with distinct capacity. Therefore, the number of labels that a node x can have at a particular iteration of the new algorithm equals the number of s–x paths which have been found until the iteration and have distinct capacity.

A label-setting algorithm without generating the transformed network explicitly, which will be referred to as LSA, is presented in Fig. 5. SLSA finds paths in the transformed network, but LSA finds paths in the original network. In LSA, a label $d(x, c)$ represents the transmission time required to send data from the source to x through a path whose capacity is c. A quickest path can be constructed recursively by the last (x^*, c^*) to be selected in LSA and $pred(x^*, c^*)$, where $pred(x^*, c^*)$ represents the node previous to x^* in the quickest path. Also, $flag(x, c) = 1$ means that at least one s–x path whose capacity is c has been found. On the contrary, $flag(x, c) = 0$ means that no s–x path whose capacity is c has yet been found. H represents a heap to store labels, and $A_G(x)$ denotes the set of arcs emanating from x in G, i.e., $A_G(x) = \{(x, y) | (x, y) \in E\}$.

The following heap operations are used in LSA (For details, see [11].):

create-heap(H): creates an empty heap.
insert($label$, H): inserts a new label into H.
find-min(H): finds and returns a label with minimum value in H.
decrease-key($value$, $label$, H): reduces the value of a label from its current value to a new $value$.
delete-min($label$, H): deletes a label with minimum value.

```
Algorithm. LSA(Label-Setting Algorithm)
1   $S \leftarrow \emptyset$
2   $flag(x,c) \leftarrow 0$ for all $x \in N$ and , $c = c_1, c_2, \ldots c_r$
3   create-heap($H$)
4   $d(s,c_r) \leftarrow 0$, $pred(s,c_r) \leftarrow 0$, $flag(s,c_r) \leftarrow 1$
5   insert($d(s,c_r), H$)
6   while $H \neq \emptyset$ do begin
7     $d(x^*,c^*) \leftarrow$find-min($H$)
8     if $x^* = t$, then stop
9     else
10      delete-min($d(x^*,c^*), H$)
11      $S \leftarrow S \cup \{(x^*,c^*)\}$
12      for each $(x^*,y) \in A_G(x^*)$ do begin
13        $\bar{c} \leftarrow \min\{c^*, c(x^*,y)\}$
14        if $x^* = s$, then $\bar{d} \leftarrow l(x^*,y) + \frac{\sigma}{\bar{c}}$
15        else, $\bar{d} \leftarrow (d(x^*,c^*) - \frac{\sigma}{c^*}) + l(x^*,y) + \frac{\sigma}{\bar{c}}$
16        end if
17        if $flag(y,\bar{c}) = 1$, then
18          if $d(y,\bar{c}) > \bar{d}$, then
19            $d(y,\bar{c}) \leftarrow \bar{d}$, $pred(y,\bar{c}) \leftarrow x^*$
20            decrease-key($\bar{d}$, $d(y,\bar{c}), H$)
21          end if
22        else
23          $d(y,\bar{c}) \leftarrow \bar{d}$, $pred(y,\bar{c}) \leftarrow x^*$, $flag(y,\bar{c}) \leftarrow 1$
24          insert($d(y,\bar{c}), H$)
25        end if
26      end for
27    end if
28  end while
```

Fig. 5. A label setting algorithm for QPP.

In LSA, lines 8 and 6 mean that LSA terminates if a quickest path is found or there remains no label in H, which implies that there exists no path from s to t. Note that s is selected as x^* at the first iteration, and then the transmission time from s to each node in $A_G(s)$ is calculated at line 14. At later iterations, transmission time to any node is calculated at line 15. If a new path from s to x is found and its capacity is not equal to any capacity value of the previously found paths from s to x, then a new label storing the transmission time of the new path will be created and inserted into the heap H(lines 22–25). Otherwise, the transmission time of the newly found path is compared with that of the previously found path with the same capacity. If the former is less than the latter, the corresponding label is set to an improved value(lines 17–21).

Let us return to the example in Fig. 1. We illustrate how LSA solves the problem as follows in Table 2:

In Table 2, the second column($d(x^*,c^*)$) represents the label obtained by find-min(H) function at each iteration of LSA. The last column represents the labels contained in H at each iteration after updating the labels of nodes reached by one of arcs in $A_G(x^*)$. The labels in H are presented in non-decreasing order for the sake of convenience.

To understand how LSA can resolve the shortcomings of SLSA, consider node 6 in Fig. 1. Since there are five distinct values in G, the transformation procedure makes five copies of node 6 in the transformed network. However, as shown in Table 2, LSA keeps only two labels which are

Table 2
An example illustrating LSA

Iteration	$d(x^*, c^*)$	S	H
Initialization	—	$\emptyset$	$\{d(1,30)\} = \{0\}$
1	$d(1,30)$	$\{(1,30)\}$	$\{d(2,30), d(3,15), d(4,8), d(5,6)\}$ $=\{18,24,40,49\}$
2	$d(2,30)$	$\{(1,30),(2,30)\}$	$\{d(3,15), d(6,30), d(4,8), d(5,6)\}$ $=\{24,28,40,49\}$
3	$d(3,15)$	$\{(1,30),(2,30),(3,15)\}$	$\{d(6,30), d(6,15), d(4,8), d(5,6)\}$ $=\{28,29,40,49\}$
4	$d(6,30)$	$\{(1,30),(2,30),(3,15),(6,30)\}$	$\{d(6,15), d(7,20), d(4,8), d(5,6)\}$ $=\{29,36,40,49\}$
5	$d(6,15)$	$\{(1,30),(2,30),(3,15),(6,30),(6,15)\}$	$\{d(7,15), d(7,20), d(4,8), d(5,6)\}$ $=\{33,36,40,49\}$
6	$d(7,15)$	Stop	—

associated with node 6: $d(6,30)$ for path $1 \to 2 \to 6$ and $d(6,15)$ for path $1 \to 3 \to 6$. In fact, there exist four paths from node 1 to node 6. The labels for path $1 \to 2 \to 6$ and path $1 \to 3 \to 6$ are created by LSA, but the labels for path $1 \to 4 \to 6$ and path $1 \to 5 \to 6$ are not created by LSA because they will not be searched until a quickest path is found. In this way, LSA can save the amount of required storage space.

4. Experimental results

We implemented LSA and compared its performance with MSA which has been known to be the most efficient [4]. For the shortest path algorithm which is repeatedly executed in MSA, we use Dijkstra's algorithm implemented with double bucket data structure. LSA was also implemented using double bucket data structure.

The generation of experimental data consists of two steps: First, we generated 20 capacitated minimum cost flow networks by running NETGEN [12]. The cost and capacity of an arc of generated networks correspond to the lead time and capacity of the arc of QPPs, respectively. Some input parameters of NETGEN need to be set before generating capacitated minimum cost flow networks. We were concerned with six parameters, and the other parameters were set to arbitrary values. The number of nodes and the number of arcs were set to values ranging from 5,000 to 60,000 and from 80,000 and 500,000, respectively. The minimum cost and maximum cost of arcs were assigned to values ranging from 10 to 10,000. Finally, the minimum capacity and maximum capacity of arcs were also set to values ranging from 10 to 10,000.

Next, we generated six QPPs from each capacitated minimum cost flow network by resetting its arcs' capacity so that the number of distinct capacity values in QPPs is 10, 20, 40, 60, 80 or 100. To reset arcs' capacity, distinct capacity values in a generated network are sorted in non-decreasing order, and then distinct capacity values are divided into the predefined number of groups. The number of groups was set to 10, 20, 40, 60, 80 or 100. Finally, the capacity of each arc of one group was reset to the maximum capacity value of the group.

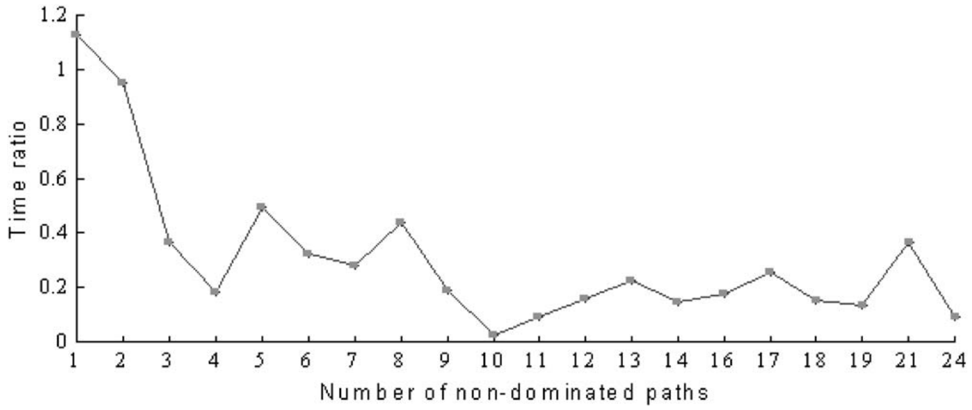

Fig. 6. Average time ratio of LSA to MSA.

Experimental results are given in Table 3. The first column(*problem*), the second column(*nodes*), and the third column(*arcs*) in Table 3 represent problem number, the number of nodes, the number of arcs, respectively. The fourth column(*sp_time*) represents the computation time that the shortest path algorithm takes to find a shortest path in each network data. The fifth column through the last column present the performance results of LSA and MSA when the number of distinct capacity values in experimental networks is 10, 20, 40, 60, 80 or 100. The fifth column is the computation time that LSA takes to find a quickest path, and the sixth one ($|S|$) is the number of labels popped from the heap H. The seventh one ($|H|$) represents the number of labels remaining in the heap H just before LSA terminated. The eighth column(*time*) is the time that MSA takes to find a quickest path, and the ninth one (*nd_paths*) is the number of non-dominated paths found by MSA algorithm. Note that the execution time of LSA and MSA excludes the preprocessing time which is spent on initializing and setting up data structure. In addition, *sp_time* is always less than the execution time of MSA even when network data have only one non-dominated path. The reason is that even when there exists only one path from the source to the destination, MSA executes the shortest path algorithm twice: At the first time, MSA finds one non-dominated path and removes some arcs whose capacity values are less than the capacity of the non-dominated path. At the second time, MSA terminates the shortest path algorithm without reaching the destination, which implies that there exists no other path between the origin and the destination.

As shown in Table 3, LSA takes less time than MSA except for some data which have only one non-dominated path. When network data have two or more non-dominated paths, LSA seems far faster than MSA. As the number of non-dominated paths becomes greater, MSA tends to take more time because the number of shortest path problems to be solved during the execution of MSA is proportional to the number of nondominated paths. Fig. 6 shows how the time ratio of LSA to MSA changes as the number of non-dominated paths increases.

The time ratio was calculated by classifying the experimental results in Table 3 according to their number of non-dominated paths and averaging the ratio of LSA's execution time to MSA's execution time of data of each group. For example, when the number of non-dominated paths is four, the time ratio is 0.18, which implies that LSA is on the average about five and a half times

Table 3
The experimental results

Problem	Nodes	Arcs	sp_time	No. of distinct capacity = 10					No. of distinct capacity = 20					No. of distinct capacity = 40				
				LSA			MSA		LSA			MSA		LSA			MSA	
				Time	$\|S\|$	$\|H\|$	Time	nd_paths	Time	$\|S\|$	$\|H\|$	Time	nd_path	Time	$\|S\|$	$\|H\|$	Time	nd_path
1	5000	80,000	0.01	0.03	741	8517	0.04	5	0.01	1043	12,099	0.06	6	0.04	1359	16,048	0.10	7
2	5000	100,000	0.00	0.03	355	5187	0.14	7	0.03	714	10,311	0.12	7	0.04	891	12,933	0.10	8
3	15,000	300,000	0.04	0.22	14,712	39,049	0.26	2	0.15	11,228	32,700	0.34	2	0.18	11,587	36,456	0.29	2
4	20,000	140,000	0.02	0.11	14,782	43,909	0.42	6	0.08	8188	36,851	0.43	9	0.06	6885	35,383	0.50	12
5	25,000	120,000	0.01	0.02	2113	7533	0.19	4	0.03	2384	8652	0.20	6	0.03	4744	16,203	0.22	7
6	30,000	300,000	0.06	0.17	20,312	41,559	0.25	2	0.18	20,455	42,940	0.25	2	0.20	20,748	45,342	0.24	2
7	30,000	500,000	0.11	0.17	10,362	59,832	0.17	1	0.20	10,357	59,785	0.17	1	0.16	10,354	59,728	0.18	1
8	40,000	400,000	0.03	0.33	37,292	54,461	0.25	2	0.33	38,289	61,808	0.24	2	0.37	39,351	69,644	0.24	2
9	40,000	400,000	0.02	0.06	4195	14,791	0.05	2	0.03	3496	12,464	0.05	2	0.04	3637	13,029	0.05	2
10	40,000	400,000	0.04	0.31	26,510	138,167	0.95	7	0.47	49,327	245,855	1.40	12	0.82	82,203	420,489	1.72	17
11	45,000	500,000	0.03	0.04	2081	19,670	0.59	5	0.03	1908	18,740	0.73	7	0.02	2060	20,270	1.02	10
12	50,000	500,000	0.01	0.04	1095	9039	0.43	3	0.04	1095	9039	0.38	4	0.03	1095	9046	0.79	5
13	50,000	500,000	0.11	0.31	28,905	87,605	1.64	7	0.34	32,689	115,834	2.62	11	0.39	38,373	160,604	2.77	14
14	50,000	500,000	0.10	0.39	44,844	144,937	1.40	8	0.44	47,925	189,450	1.98	13	0.41	47,034	214,110	2.34	16
15	55,000	450,000	0.09	0.26	34,291	60,041	0.34	2	0.28	34,513	61,887	0.34	2	0.31	34,855	64,220	0.33	2
16	60,000	300,000	0.04	0.08	9291	29,439	0.20	3	0.08	9878	31,896	0.20	3	0.08	10,238	33,339	0.19	3
17	60,000	350,000	0.13	0.25	35,879	119,267	0.72	3	0.42	52,110	176,013	0.85	5	0.59	81,093	246,114	0.87	5
18	60,000	400,000	0.06	0.28	36,388	90,143	0.34	2	0.32	37,198	94,999	0.34	2	0.30	38,200	99,337	0.35	2
19	60,000	400,000	0.09	0.18	21,624	55,799	0.54	2	0.21	21,637	55,877	0.54	2	0.18	21,676	56,155	0.88	3
20	60,000	500,000	0.12	0.23	22,789	117,850	0.68	4	0.26	23,239	135,940	0.91	6	0.24	23,321	145,232	1.11	6

Table 3 (*Continued*).

Problem	No. of distinct capacity = 60					No. of distinct capacity = 80					No. of distinct capacity = 100				
	LSA			MSA		LSA			MSA		LSA			MSA	
	Time	$\|S\|$	$\|H\|$	Time	nd_paths	Time	$\|S\|$	$\|H\|$	Time	nd_paths	Time	$\|S\|$	$\|H\|$	Time	nd_paths
1	0.05	1610	19,073	0.08	8	0.05	1732	20,395	0.09	8	0.03	1857	22,063	0.08	8
2	0.03	820	12,234	0.10	8	0.04	845	12,586	0.09	8	0.04	859	12,874	0.08	8
3	0.18	12,125	40,336	0.25	2	0.19	12,413	42,734	0.26	2	0.21	12,665	44,426	0.25	2
4	0.08	7707	40,405	0.56	12	0.09	7384	39,364	0.55	12	0.10	7503	40,646	0.44	12
5	0.04	7312	23,782	0.22	7	0.06	12,407	33,345	0.23	7	0.07	9909	26,602	0.21	7
6	0.22	20,955	47,092	0.25	2	0.23	21,068	48,006	0.24	2	0.23	21,147	48,561	0.24	2
7	0.21	10,354	59,729	0.18	1	0.20	10,354	59,728	0.17	1	0.23	10,354	59,728	0.17	1
8	0.40	39,962	73,767	0.24	2	0.38	40,275	75,842	0.24	2	0.45	40,496	77,363	0.24	2
9	0.04	3735	13,432	0.05	2	0.04	3821	13,741	0.05	2	0.07	3879	13,957	0.05	2
10	0.19	16,417	129,288	2.21	19	0.24	18,597	146,688	2.32	21	0.21	20,229	160,235	2.30	24
11	0.06	2132	21,235	1.17	11	0.06	2105	20,957	1.55	12	0.09	2121	21,222	1.59	12
12	0.04	1097	9072	1.63	6	0.02	1098	9091	1.63	6	0.06	1108	9210	1.66	7
13	0.38	35,220	130,152	3.36	18	0.26	24,486	90,483	2.80	17	0.46	40,485	146,373	2.83	18
14	0.44	46,867	222,577	2.43	17	0.49	46,971	226,955	2.72	18	0.47	47,109	229,746	2.59	19
15	0.33	35,136	65,942	0.34	2	0.32	35,303	67,011	0.33	2	0.35	35,411	67,670	0.33	2
16	0.11	10,360	33,799	0.20	3	0.12	10,450	34,099	0.19	3	0.11	10,535	34,477	0.21	3
17	0.97	103,866	284,328	1.02	5	1.47	116,037	302,646	1.00	6	1.23	129,429	305,521	1.26	7
18	0.33	38,790	101,769	0.33	2	0.35	39,187	103,409	0.35	2	0.39	39,480	104,575	0.34	2
19	0.22	21,688	56,237	0.87	3	0.19	21,691	56,245	0.83	3	0.20	21,695	56,301	0.69	3
20	0.37	23,633	151,195	1.01	7	0.28	23,847	154,156	1.17	7	0.33	24,568	159,986	1.08	6

faster than MSA. In Fig. 6, we find that as the number of non-dominated paths increases, the time ratio tends to decrease.

In addition, $|S|$ is relatively small compared to the number of $|H|$. This indicates that only a small portion of labels are permanently labelled. The total number of labels, $(|S| + |H|)$, appeared far less than the worst-case number of labels, rn. This implies that the storage space requirement of LSA is not too restrictive.

5. Conclusion

In this paper, we proposed a label-setting algorithm for finding a quickest path when the amount of data to be transmitted is given. The proposed algorithm is based on the important property, that each subpath of a quickest path is also a quickest path, which holds in a transformed network. The proposed algorithm calculates the transmission time from the origin to intermediate nodes, which is utilized to avoid enumerating non-dominated paths whose transmission time is far greater than the minimum transmission time. Although the computational complexity of the proposed algorithm is the same as that of existing algorithms, experimental results showed that it is more efficient than existing algorithms when a network has two or more non-dominated paths.

Acknowledgements

We thank an anonymous referee for his helpful comments. This work was supported by Korea Science and Engineering Foundation (KOSEF) under Grant No. 2000-1-31500-001-02 and R01-2002-000-00168-0.

References

[1] Chen YL, Chin YH. The quickest path problem. Computers and Operations Research 1990;17:153–61.
[2] Moore MH. On the fastest route for convoy-type traffic in flowrate-constrained networks. Transportation Science 1976;10:113–24.
[3] Rosen JB, Sun S-Z, Xue GL. Algorithms for the quickest path problem and the enumeration of quickest path. Computers and Operations Research 1991;18:579–84.
[4] Martins EQV, Santos JL. An algorithm for the quickest path problem. Operations Research Letters 1997;20:195–8.
[5] Chen G-H, Hung YC. On the quickest path problem. Information Processing Letters 1993;46:125–8.
[6] Chen YL. Finding the k quickest simple paths in a network. Information Processing Letters 1994;50:89–92.
[7] Kagaris D, Pantziou GE, Tragoudas S, Zaroliagis CD. Transmissions in a network with capacities and delays. Networks 1999;33:167–74.
[8] Xue G, Sun S, Rosen JB. Fast data transmission and maximal dynamic flow. Information Processing Letters 1998;66:127–32.
[9] Lin Y-K. Extend the quickest path problem to the system reliability evaluation for a stochastic-flow network. Computers and Operation Research 2003;30:567–75.
[10] Hu TC. Combinatorial algorithms. Reading, MA: Addison-Welsey, 1982.
[11] Ahuja RK, Magnanti TL, Orlin JB. Networks flows: theory, algorithms and applications. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[12] Klingman D, Napier A, Stutz J. NETGEN: a program for generating large scale capacitated assignment, transportation, and minimum cost flow networks. Management Science 1974;20:814–20.

Chan-Kyoo Park currently works in Department of IT Audit and Supervision at National Computerization Agency, Korea. He received his Ph.D. in operations research from Seoul National University. His research interests are in mathematical programming and its applications to data mining.

Sangwook Lee is a Ph.D. candidate in Department of Industrial Engineering at Seoul National University, Korea. His research areas include linear, nonlinear and integer programming, network theory and computer applications.

Soondal Park is a professor of Department of Industrial Engineering at Seoul National University, Korea. He received his Ph.D. in Mathematics from University of Cincinnati. His research interests include deterministic operations research and its computer applications. He is the author of LP programs, LPAKO, LPABO and LPASO, and of twenty-five books on various fields including operations research.