Journal of the Operations Research
Society of Japan
Vol. 27, No. 1, March 1984

PRACTICAL EFFICIENCIES OF EXISTING SHORTEST-PATH ALGORITHMS AND A NEW BUCKET ALGORITHM

Hiroshi Imai
University of Tokyo

Masao Iri
University of Tokyo

(Received May 31, 1983: Revised December 8, 1983)

Abstract For the problem of finding the shortest paths from a prescribed vertex to the other vertices in a given network with nonnegative arc lengths, we investigate practical efficiencies of typical existing algorithms, i.e., the label-correcting method with a FIFO and that of a two-way sequence, and the label-setting method with a heap and that with a 1- or 2-level bucket system. We propose also a new method with a variable bucket system and compare it with those existing methods. Among the existing methods, the label-correcting method of a two-way sequence and the label-setting method with a 1-level bucket system have been found efficient in most cases. The new proposed method is not only as efficient as those but it is also robust for a large variety of networks, so that it may be recommended for practical use.

Introduction

The shortest-path problem is one of the theoretically most fundamental and practically most important problems in network flow theory, and various algorithms have been developed since the latter half of 1950's. From the theoretical point of view, it is known that, for the problem of finding the shortest paths from a prescribed vertex (called the "entrance") to the other vertices in a dense network with nonnegative arc lengths, Dijkstra's method [4] is the most efficient, and that, for the problem of finding the shortest paths of all pairs of vertices in a dense network where arc lengths may be negative so long as there is no negative cycle, Floyd and Warshall's method [5], [10] is the best. For the problem in a sparse network, many variants of Dijkstra's method employing sophisticated data-structures such as a heap and other priority queues have been developed [7] in order to improve the theoretical worst-case time bound.

From the practical computational point of view, on the other hand, studies have been made on computational estimation and improvement of the shortest-path

algorithms, and several useful results have been obtained. Algorithms studied in this way may roughly be classified into the label-correcting method (potential method) and the label-setting method (Dijkstra's method). In [2], [3], it was shown that (i) the two-way sequence method [9], which is one of the variants of the label-correcting method, is very efficient; especially for traffic road networks, it is the most effective, and (ii) among the variants of the label-setting method, the methods using buckets are efficient.

In this paper, we consider the shortest-path problem from an entrance to all the other vertices in a network where arc lengths are nonnegative. (It should be noted that the assumption of the nonnegativity of arc lengths is not so restrictive since, if some arcs have negative length, we can quickly transform the problem to that with nonnegative arc lengths by a preprocessing in most cases of practical interest.) We investigate the practical efficiencies of the algorithms which have been proposed so far by means of computational experiments. The algorithms examined here are the FIFO method and the two-way sequence method of the label-correcting type and the heap method and the 1- and 2-level bucket methods of the label-setting type and a new variable bucket method which we propose in this paper in order to use buckets more effectively. We examine these methods for several types of networks such as random networks, grid networks and real road networks of the Tokyo metropolitan area, the Kofu city and the Kanto district of Japan. We not only employ those road networks as they are but also use those networks with some of arc lengths modified. Starting from a real road network and modifying the topology and/or arc lengths to some extent will be a powerful technique for generating many test networks which are suitable for evaluating the practical efficiencies of network algorithms by computational experiments.

1. Labelling Methods for the Shortest-Path Problem

1.1. Definitions

Let $G=(V,A)$ be a directed graph with vertex set V and arc set A. For each arc $a\in A$, we denote by $\partial^+ a$ (resp. $\partial^- a$) its initial (resp. terminal) vertex, and for vertex $v\in V$, we denote by $\delta^+ v$ (resp. $\delta^- v$) the set of arcs going out of (resp. coming into) it. On G, a *length function* d: $A\to\mathbf{R}_+$ ($\mathbf{R}_+$: nonnegative reals) and a special vertex s (called the *entrance* or *source*) are given. Then, we call $N=(V,A,\partial^+,\partial^-,d,s)$ a *network* and G as the *underlying graph* of N.

A *directed path* from vertex u to vertex v is a sequence of arcs $P=a_0,a_1,\ldots,a_k$ such that $\partial^+ a_0=u$, $\partial^- a_k=v$ and $\partial^- a_{i-1}=\partial^+ a_i$ $(i=1,\ldots,k)$. The *length* of path P is defined to be $\sum_{i=0}^{k} d(a_i)$. In the following, we shall consider the single-entrance prob-

lem, i.e., the problem of finding the shortest paths from the entrance s to all the other vertices in network N.

1.2. Basic data-structure for representing a network

For representing a network N in a computer, we shall essentially use the "standard data-structures" [6]. The underlying graph $G=(V,A)$ of N is represented by functions ∂^- and δ^+. The function ∂^- is represented by an array of size $|A|$. For each vertex $v\in V$, δ^+v is expressed as a one-way list having a head pointer, so that, in total, the function δ^+ is represented by an array of size $|A|$ for the lists and an array of size $|V|$ for the head pointers. The length function d is represented by an array of size $|A|$. Thus the network N is represented by three arrays of size $|A|$ and an array of size $|V|$.

1.3. Labelling methods

A *label* $(a,p(v))$ attached to vertex v indicates that a path has been found from s to v such that its length is $p(v)$ and the last arc on this path is a $(\partial^-a=v)$. We call $p(v)$ the *potential* of vertex v. Initially we set the labels on the vertices except s to be $(*,\infty)$ where "*" means "being undefined" and an infinite potential means that no path has been found from s to the vertex. The labels obtained at the end of the computation of the labelling method completely define the shortest paths which have been obtained, in the sense that those paths can be readily constructed in an obvious way with the aid of those labels.

A labelling method consists of two operations, i.e. the operation of *scanning arcs* and that of *searching vertices*. By scanning an arc $a=(u,v)\in A$ $(u=\partial^+a,\ v=\partial^-a)$, we mean checking whether the inequality $p(v)>p(u)+d(a)$ holds or not, and, if it holds, modifying the label of vertex v into $(a,p(u)+d(a))$. By searching vertex $v\in V$, we mean scanning each arc in δ^+v. According to the manner of searching, the labelling methods can be classified into two: the method of *label-correcting type* (or, *potential methods*) and that of *label-setting type* (or, *Dijkstra-type methods*), which are outlined below.

(1) Label-correcting method

The vertices with label $(*,\infty)$ are said to be "unlabelled" whereas the other vertices are said to be "labelled".

(step 1) Give the special label $(*,0)$ to s, and the label $(*,\infty)$ to all the other vertices. All the vertices are defined to be "unsearched".

(step 2) If there is no "labelled and unsearched" vertex, then halt {*the shortest paths have been found*}; otherwise, take such a vertex v and go to step 3.

(step 3) Search vertex v; that is, for each arc a in δ^+v, if $p(\partial^-a)>p(v)+d(a)$, then give vertex ∂^-a the label $(a,p(v)+d(a))$, and make vertex ∂^-a unsearched. Having searched v, make v "searched", and return to step 2.

A label-correcting method works correctly even when there is an arc of negative length so long as there is no negative cycle.

(2) Label-setting method

Labels are classified into three: the labels with infinite potentials, the *tentative labels* and the *permanent labels,* where the potentials of the tentative labels and the permanent ones are finite. A permanent label is attached to vertex v iff the path from s to v indicated by the label on v has already been found to be the shortest, and will never change any further. A tentative label, on the other hand, may become permanent with the current value of its potential or may have its value modified in the subsequent computation.

(step 1) Attach the special "tentative" label $(*,0)$ to s, and give all the other vertices label $(*,\infty)$.

(step 2) If all the vertices already have permanent labels, then halt {*the shortest paths have been found*}; otherwise, take a vertex v with a tentative label which has the smallest potential among the vertices with tentative labels, and go to step 3.

(step 3) Make the label on v "permanent" and search vertex v: that is, for each arc a in δ^+v, if $p(\partial^- a)>p(v)+d(a)$, give the vertex $\partial^- a$ the tentative label $(a,p(v)+d(a))$. Having searched v, return to step 2.

A label-setting method works correctly if arc lengths are nonnegative. Note that, when we want to find only the shortest path from s to another prescribed vertex t (called the "exit" or "sink"), we may terminate the computation as soon as the label attached to t is made permanent.

The efficiency of an algorithm or a method depends substantially upon the implementation, i.e., upon how to determine the vertex to be next searched, so that we shall discuss efficient implementations of label-correcting methods and of label-setting methods in the following two sections.

2. Label-Correcting Methods

The efficiency of a label-correcting method depends upon the order of searching "labelled and unsearched" vertices. The following two methods are known to be efficient.

(1) FIFO method

This method maintains the set of "labelled and unsearched" vertices with a FIFO list (first-in first-out list; or "queue"). This list may be implemented as a one-way list with the head and the tail pointers on an array of size $|V|$. This method can find the shortest paths in $O(|A|\cdot|V|)$ time even in the worst case.

(2) Two-way sequence method

This method maintains the set of "labelled and unsearched" vertices by a one-way list having the head and the tail pointers in the following way: When a vertex having the infinite potential has the potential modified to a finite value, it is inserted to the tail of the list, and when a vertex having a finite potential has the potential updated, it is inserted to the head if it is not contained in the current list (if it is contained in the current list, we do nothing). At step 2 of label correction, the head vertex of the list is taken out and away from the list and searched [9]. This method works efficiently in most practical cases, but its theoretical worst-case bound is proved to be non-polynomial in $|V|$ and $|A|$ [8].

3. Label-Setting Methods

A label-setting method generally finds the shortest paths in $O(|V|^2)$ time in the worst case. The efficiency of a label-setting method depends upon how to find a vertex with a tentative label which has the smallest potential among the vertices with tentative labels. The following methods have been developed for implementing the label-setting method efficiently.

(1) Heap method

Maintaining the set of vertices with tentative labels by a priority queue [1], we can find the shortest paths in $O(|A| \log |V|)$ time [7], where a heap is ordinarily employed as a priority queue.

(2) 1-level bucket method [2]

Let m and M be the minimum and the maximum, respectively, of the lengths of arcs in the network, where m is assumed to be positive. Set $x=\lceil M/m \rceil+1$, where $\lceil z \rceil$ denotes the smallest integer that is not smaller than z. Then, the length of each shortest path from s is at most $M(|V|-1)$. The interval from 0 to $M(|V|-1)$ is divided into $x(|V|-1)$ subintervals of width m. The 1-level bucket method maintains the set of vertices with tentative labels in x buckets according to their potentials as follows.

A vertex v with a tentative label $(a,p(v))$ is put in the kth bucket such that $km \leqslant p(v) < (k+1)m$. A set of vertices in the same bucket is linked with one another by a two-way list having a head pointer. In order to find the vertex of the minimum-potential tentative label, we have only to find the smallest-numbered bucket that is nonempty (a bucket is nonempty if it contains a vertex). Since the bucket width is m, all the tentative labels of the vertices in the smallest-numbered nonempty bucket can be made permanent simultaneously. Moreover, when the potential of a vertex is updated so that it is to be moved to another bucket, we can perform the necessary modification of the two-way lists of the relevant buckets in a constant time. Thus the

shortest paths can be found in $O(|A|+|V|x)$ time.

The number of buckets necessary for the computation is at most $x(|V|-1)$, which is ordinarily prohibitively large. However, noting that the difference of the maximum and the minimum of potentials of tentative labels at any stage of computation is at most M, we can do everything with only x buckets of width m by putting vertex v with potential $p(v)$ in the $(\lfloor p(v)/m \rfloor \bmod x)$th bucket. This technique reduces the total space requirement to $O(|A|+x)$.

(3) 2-level bucket method [2]

The efficiency of the 1-level bucket method depends highly upon the parameter x. In order to mitigate the dependence upon x, generalization to the 2-level bucket method, and more generally the k-level bucket method, has been proposed. The 2-level bucket method maintains the set of vertices with tentative labels by a 2-level bucket system. That is, on the first level, the vertices with tentative labels are distributed into $\lceil\sqrt{x}\rceil$ buckets of width $\lceil\sqrt{x}\rceil m$, and, on the second level, the vertices which are contained in the smallest-numbered bucket that is nonempty on the first level are distributed into $\lceil\sqrt{x}\rceil$ buckets of width m of the second level. By doing so, we can reduce the total complexity of the method to $O(|A|+|V|\sqrt{x})$ in time and $O(|A|+\sqrt{x})$ in space.

More generally, from the theoretical point of view, we may introduce the k-level bucket method whose time bound is $O(|A|+k|V|x^{1/k})$. But the larger the number of levels k is, the greater will be the overhead for moving between consecutive levels in the bucket system, which would make the method practically inefficient for large k.

4. Variable Bucket Method

The bucket methods such as the 1- and 2-level bucket methods have the disadvantage that the necessary computation time and memory space are not only bounded by the topology of the network but the efficiency substantially depends also upon the lengths of arcs in the network. Especially, the 1-level bucket method is quite inefficient in the case where there is a very short arc and, at the same time, a very long shortest path.

So, we will propose here another way of utilizing buckets. The new method uses buckets whose widths vary according to the circumstances but whose number is fixed a priori by the topology of the network. We will call it the *variable bucket method*. The following is a brief description of the algorithm of the proposed method.

Suppose that there are k vertices with tentative labels at a stage and that their potentials range from $t_{\min}$ to $t_{\max}$. Among those k vertices, we maintain the vertices whose potentials are between $t_{\min}$ and t (a threshold value) by buckets of width w, where t and w are defined, for given parameters α and β, by

(4.1) $t=(1-\alpha)t_{\min}+\alpha t_{\max} \quad (\alpha>0)$,

(4.2) $w=\max\{m, \beta(t-t_{\min})/k\}$.

If the width w of buckets is greater than m, we cannot change the tentative labels of all the vertices in the top (i.e., smallest-numbered) bucket that is nonempty into permanent ones simultaneously, but we have to apply further a label-correcting or label-setting method to those vertices. When there appear, during the course of computation, vertices with tentative labels whose potentials are greater than t, they are kept in a list which we call the "*overflow-list*". When all the labels with potentials less than t become permanent, i.e., when all the vertices with tentative labels are in the overflow-list, we reconstruct the buckets with respect to the number k' of vertices in the overflow-list, and the maximum $t'_{\max}$ and the minimum $t'_{\min}$ of potentials of those vertices (i.e., we set $k:=k'$, $t_{\max}:=t'_{\max}$ and $t_{\min}:=t'_{\min}$), and iterate the above procedure until all the vertices come to have permanent labels. For the starting values at the initial stage, we may take $k=1$ and $t_{\min}=t_{\max}=0$. Concerning parameters α and β, the computational experiments show that the choice $\alpha=\beta=1$ seems good. (Note that the efficiency of this algorithm is not so sensitive to the choice of these parameters; see the following section.)

In the case of $\alpha\geqslant 1$, the total number of buckets which the proposed variable bucket method requires is at most $|V|/\beta$, and is independent of the lengths of arcs, which is not the case with the 1- and 2-level bucket methods. It is not difficult to see that the theoretical worst-case time bound of the variable bucket method coincides with that of the method which we employ for manipulating the set of vertices in the top nonempty bucket. For example, if a label-correcting method is employed for that purpose, the variable bucket method can be regarded as a method which, so to speak, suppresses the instability of the label-correcting method with a slight extra work. If a label-setting method is employed, the variable bucket method speeds up finding the minimum-potential vertex by means of buckets.

5. Computational Experiments

5.1. Design of computational experiments

In this section, we evaluate the computational efficiencies of the shortest-path algorithms described above by computational experiments. Here, we have no problem in coding the algorithms practically, since, unlike other network-flow algorithms, it is almost clear, from the descriptions of those algorithms, how to implement them and what data-structure should be adopted. The algorithms are so simple that we may code them without using any subroutine in the main part of the program to avoid nesting subroutines which usually affects the running time seriously.

Table 5.1. Memory space required by the tested algorithms: $c_v|V|+c_a|A|+c_1x+c_2\sqrt{x}$

$$\left(x=\frac{\text{the maximum of arc lengths}}{\text{the minimum of arc lengths}}\right)$$

	c_v	c_a	c_1	c_2
FIFO	4	3		
TWSQ	4	3		
HEAP	5	3		
B1	5	3	1	
B2	5	3		2
BV	7	3		

Thus, in the evaluation of the computational efficiency, the question is what types of test networks we should employ. For test networks, we first used networks with random graphs and grid networks. Although random networks are far from real networks, they, especially if they are dense, are expected to share the same fundamental properties with complete networks and, hence, to reflect the theoretical complexity of each algorithm in the worst case on computational results. Grid networks may be considered as models of real road networks in some cities. In addition to the above artificial networks, we took up the real road networks of the Tokyo metropolitan area, the Kofu city and the Kanto district of Japan, and recast them in various ways. Specifically, we assigned to the road network of the Kanto district different types of length functions. We believe that the networks constructed in this way from real road networks are suitable for use in evaluating the practical efficiencies of network algorithms. For the variable bucket method, we tested the effect of parameters α and β on the efficiency, too.

The tested algorithms are:

- label-correcting methods
  - FIFO method (FIFO)
  - two-way sequence method (TWSQ)
- label-setting methods
  - heap method (HEAP)
  - 1-level bucket method (B1)
  - 2-level bucket method (B2)
- variable bucket method (BV)

Each method will be referred to by the abbreviation indicated above. In the variable bucket method, we employed the two-way sequence method for vertices in the same bucket. The memory space required by each code is listed in Table 5.1. All the

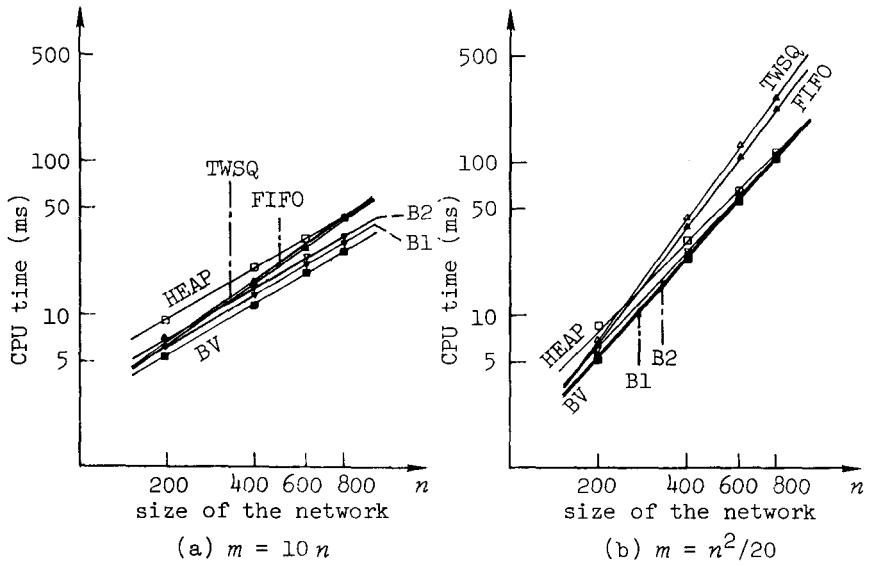

Fig.5.1. Running times for random networks (n vertices and m arcs)

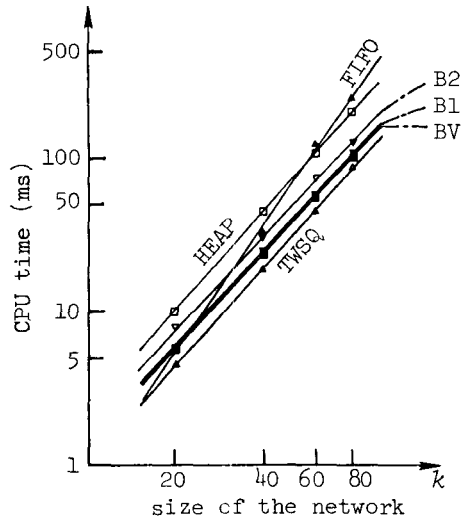

Fig.5.2. Running times for $k \times k$ grid networks (k^2 vertices)

methods were coded and implemented by FORTRAN77 (opt=2) on HITAC M-280H (VOS3/JSS4) of the Computer Centre of the University of Tokyo. The running time was measured by subroutine CLOCK which gives a time exact up to about one millisecond excluding the time for input and output.

5.2. Computational results

(1) Random networks

Random network $N(n,m,c)$ was constructed as follows where n, m and c are parameters. Initially, n vertices are given, and then m arcs are selected randomly

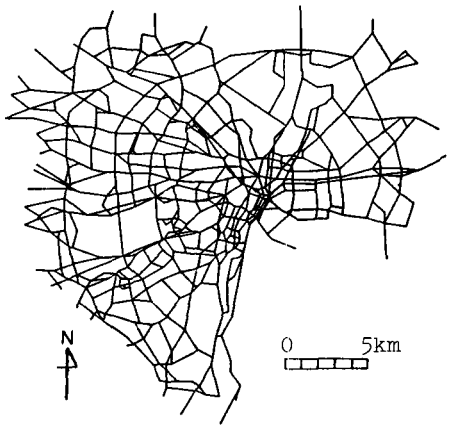

Fig.5.3. Road network of the Tokyo metropolitan area (513 vertices and 849 arcs)

Fig.5.4. Road network of the Kofu city area (1173 vertices and 1820 arcs)

from among all possible $n(n-1)$ arcs; the length of an arc is a random integer between 1 to c, and entrance s is chosen randomly from among the n vertices.

We set $c=100$, and examined six cases with $m=10n$ and $m=n^2/20$ for $n=200$, 400, 800. For each case, we solved twenty five different problems (i.e., the problems for twenty five different random networks). The average running times were as depicted in Fig.5.1.

(2) Grid networks

A $k\times k$ square grid network has k^2 vertices and about $4k^2$ arcs. Entrance s is randomly chosen and the length of an arc is also a random integer between 1 to c.

We set $c=100$, and considered four cases of $k=20$, 40, 60, 80. For each case, we solved twenty five different problems. The average running times were as shown in Fig.5.2.

(3) Real road networks

The road network of the Tokyo metropolitan area, shown in Fig.5.3, has 513 vertices and 849 arcs, and the lengths of arcs range from 10 to 694 (1 unit = 10m). The road network of the Kofu city, shown in Fig.5.4, has 1173 vertices and 1820 arcs, and the lengths of arcs range from 5 to 328 (1 unit = 10m). In each network, we solved

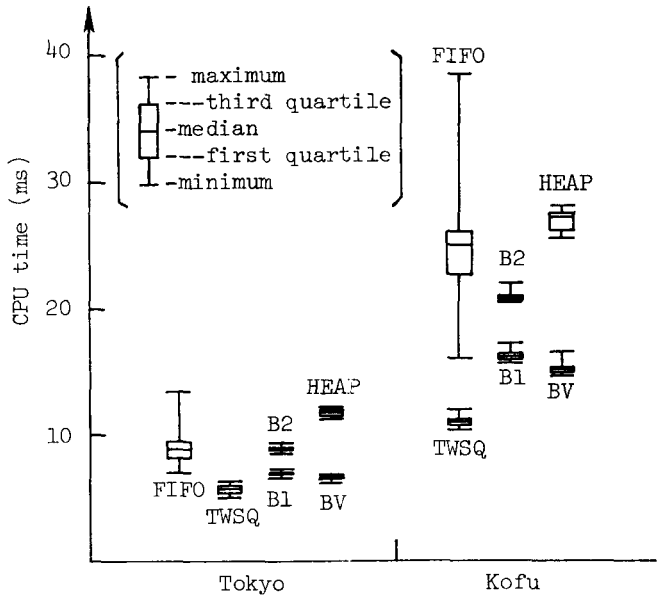

Fig.5.5. Running times for the road networks of the Tokyo metropolitan area and the Kofu city area

twenty different problems, in each of which entrance s was randomly determined. The distribution of the running times were as shown in Fig.5.5.

The road network of the Kanto district, shown in Fig.5.6, has 6922 vertices and 10112 arcs, and the lengths of arcs is from 37m to 31506m, where the arcs longer than 5km are 296 in number. In this network, we considered the following four types of problems which are different from one another in the lengths of arcs.

(A) The real road lengths were adopted as they were.
(B) The lengths of the above-mentioned 296 long arcs were multiplied by five.
(C) Random integers between 1 to 100 were assigned to the arcs for their lengths.
(D) Random integers between 1 to 10000 were assigned to the arcs for their lengths.

For each case, we solved twenty different problems with randomly chosen entrances. The distribution of the running time were as shown in Fig.5.7. Since FIFO were already found to be quite inefficient for road networks, and it really took much time, we omitted it from Fig.5.7. In case (A), we examined the effect of parameters α and β on the efficiency of the variable bucket method. The results were as shown in Fig.5.8.

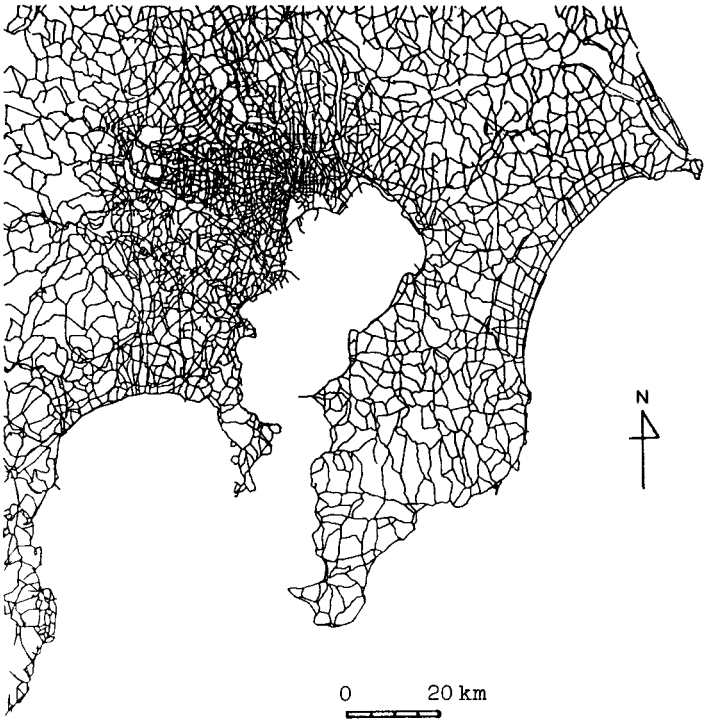

Fig.5.6. Road network of the Kanto district
(6922 vertices and 10112 arcs)

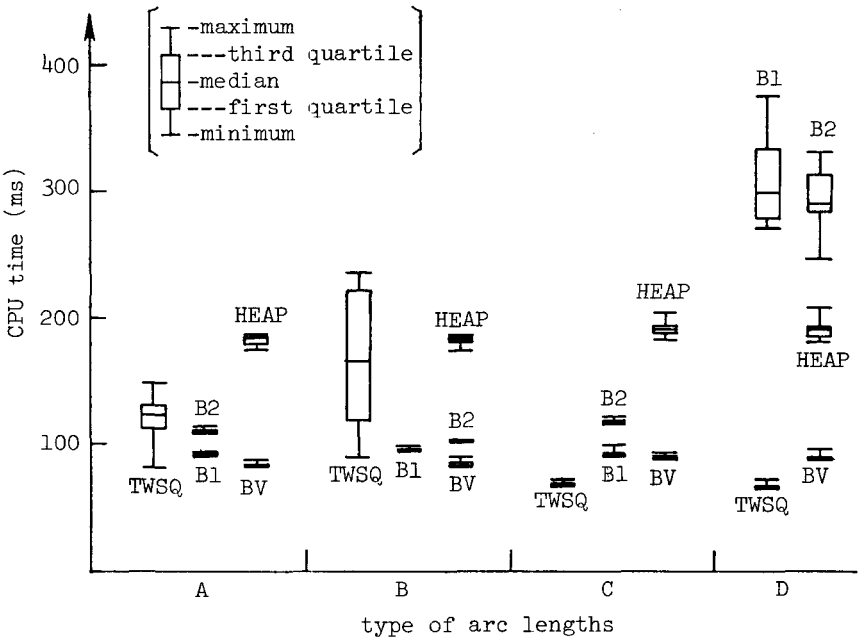

Fig.5.7. Running times for the road network of the Kanto district

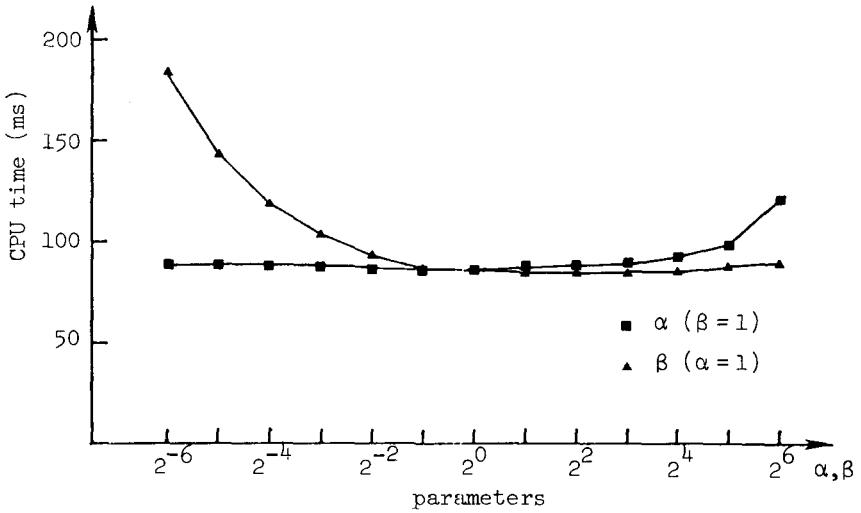

Fig.5.8. Effect of the parameters of the variable bucket method on the running time

5.3. Evaluation of the computational results

Label-correcting methods are generally not efficient for random networks. For grid networks and road networks, the performance of the FIFO method is poor, but the two-way sequence method works relatively efficiently. Especially, for grid networks and rather well arranged road networks such as that of the Kofu city (which is quite similar to a grid network), the two-way sequence method is the best. However, as is seen from the computational results of cases (A) and (B) for the road network of the Kanto district, the two-way sequence method will become less efficient for networks of complicated topological structure with highly nonuniform arc lengths.

Generally speaking, the label-setting methods are efficient for random networks, but take a bit more time for grid and small road networks than the two-way sequence method. Among the label-setting methods, the heap method is robust and stable but the nearly slowest probably because it takes too much time to maintain the heap. The 1-level bucket method is in most cases as fast as and more stable than the two-way sequence method. The 2-level bucket method usually takes more time than the 1-level bucket method. These two bucket methods become badly inefficient for some length functions, as is shown in cases (C) and (D) for the Kanto road network.

The variable bucket method, as is seen from the computational results for the Kanto district, is robust against the change of structures and length functions of the network and belongs to the fastest family in all cases. It is also seen that the efficiency of the variable bucket method is not so sensitive to parameters α and β. Note that, as α and β grow larger, the variable bucket method will tend to resemble the method used for manipulating the vertices in the same bucket (which is the two-

way sequence method in the present experiment) and, as β gets smaller, it will tend to resemble the 1-level bucket method.

6. Conclusion

It may be said that there is little room to improve any further the efficiency of the shortest-path algorithms dramatically so long as we adopt the present model of computation. As the efficient shortest-path algorithms for practical purposes, either the two-way sequence method, the 1-level bucket method or the variable bucket method is recommended according to the circumstances. For typical and simple road networks, the two-way sequence method is good, though this method is less efficient for complicated networks, and is quite inefficient for random networks and dense networks. So long as the ratio of the maximum and the minimum of the lengths of arcs is small and the maximum of the lengths of shortest paths is small, the 1-level bucket method is as efficient as and robuster than the two-way sequence method, displaying merits of the label-setting method. The variable bucket method is very robust and efficient on the average. In fact, for all the problems we tested, it is ranked the best or, if not the best, the next best. When a label-correcting method such as the two-way sequence method is employed for manipulating vertices in the same bucket, the variable bucket method will have both the merits of the bucket method of the label-setting type and those of the label-correcting method united.

References

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D.: *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

[2] Denardo, E. V., and Fox, B. L.: Shortest-Route Methods: 1. Reaching, Pruning, and Buckets. *Operations Research,* Vol.27, No.1 (1979), 161-186.

[3] Dial, R., Glover, F., Karney, D., and Klingman, D.: A Computational Analysis of Alternative Algorithms and Labelling Techniques for Finding Shortest Path Trees. *Networks,* Vol.9, No.3 (1979), 215-248.

[4] Dijkstra, E. W.: A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik,* Vol.1 (1959), 269-271.

[5] Floyd, R. W.: Algorithm 97: Shortest Path. *Communications of the ACM,* Vol.5, No.6 (1962), 345.

[6] Iri, M., et al.: Fundamental Studies of the Techniques for Processing Network Problems in Operations Research by Computers (in Japanese). *Technical Report T-73-1,* Operations Research Society of Japan, 1973.

[7] Johnson, E. L.: On Shortest Paths and Sorting. *Proceedings of the 25th ACM*

National Conference, 1972, 510-517.

[8] Kershenbaum, A.: A Note on Finding Shortest Path Trees. *Networks,* Vol.11, No.4 (1981), 399-400.

[9] Pape, U.: Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem. *Mathematical Programming,* Vol.7 (1974), 212-222.

[10] Warshall, S.: A Theorem on Boolean Matrices. *Journal of the Association for Computing Machinery,* Vol.9 (1962), 11-12.

Hiroshi IMAI: Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Hongo, Bunkyo-ku,Tokyo, Japan 113