DOI 10.1007/s10559-014-9666-9
Cybernetics and Systems Analysis, Vol. 50, No. 5, September, 2014

A TWO-CRITERION LEXICOGRAPHIC ALGORITHM FOR FINDING ALL SHORTEST PATHS IN NETWORKS

V. A. Vasyanin

UDC 519.168

Abstract. *An algorithm for finding all shortest paths in an undirected network is considered. The following two criteria are used: the minimum number of arcs in a path and a minimum path length. The algorithm is analyzed for complexity, and it is empirically shown that, with increasing the network density, its computational efficiency becomes higher than that of the Floyd algorithm adequately modified to find the shortest path with the use of a two-step criterion.*

Keywords: *multicriteria problem of finding shortest paths, algorithm, computational complexity.*

INTRODUCTION

In many theoretical and applied optimization problems on graphs such as, for example, the design of transport, information, and telecommunication networks, shortest paths (SPs) should be found. The majority of publications devoted to questions of finding SPs consider problems of finding SPs with the use of one criterion, namely, a minimum or a maximum of the sum of lengths of arcs in a path. The length of an arc is understood to be some of its parameter, for example, the distance along the arc, expenditures for the transportation along the arc, throughput of the arc, etc.

Along with problems of finding SPs with the use of one criterion, of considerable interest are multicriteria shortest path problems (MSPs) naturally arising in many applications when there are some different parameters characterizing nodes and arcs of a network. In the majority of cases, the choice of optimal shortest paths according to extreme values of these parameters is a complicated problem since parameters can be inconsistent and can compete among themselves. In contrast to the problem of optimization of paths with respect to one criterion, the conception of optimization of a MSP problem on the whole consists of the necessity of finding some optimal solution with respect to all given criteria. In this case, as a rule, there is not one but a set of Pareto-optimal solutions of the problem that satisfy all criteria.

Since a review of modern multicriteria algorithms is a topic in its own right and is beyond the scope this article, we only note that they were investigated less thoroughly, for example, P. Hansen published [1] in 1979 and devoted it to systematic investigation of two-criterion problems on paths. P. Hansen [1] and, independently, A. Warburton [2] have developed fully polynomial time approximation schemes (FPTASs) to solve the MSP problem with obtaining Pareto-optimal solutions in the case of two criteria. Two-criterion problems were also investigated rather well in [3]. After these publications, many works were devoted to multicriteria problems on SPs in which issues of developing efficient algorithms of finding Pareto-optimal paths with the use of several competitive criteria were investigated. For the cases when the number of optimization criteria exceeds two, any special results are not obtained yet. Judging by references and publications on the Internet, any special advancements in solving MSP problems with the use of FPTASs are not observed yet.

This article considers algorithms for finding SPs that are lexicographically ordered and satisfy two criteria. It is well known that these algorithms have polynomial complexity in contrast to algorithms finding Pareto-optimal shortest paths. Some algorithms for finding paths with the use of a two-step criterion were considered in [4, 5]. The idea of the algorithm

Institute for Telecommunications and Global Information Space, National Academy of Sciences of Ukraine, Kyiv, Ukraine, *archukr@meta.ua*. Translated from Kibernetika i Sistemnyi Analiz, No. 5, pp. 122–131, September–October, 2014. Original article submitted January 30, 2014.

proposed in [4] is close to the idea of the Moore algorithm [6], and its distinctive feature is the implementation of the procedure of arrangement of "marks" with the help of logical operations over rows of matrices characterizing a network. As is shown in [7], the asymptotic complexity of algorithms of this kind amounts to $O(n^3)$, where n is the number of nodes in a network. In [5], an algorithm is described that finds shortest path trees with the use of the following two-step criterion: a minimum rank of a path and, if ranks are equal, then a maximum throughput (the rank of a path is understood to be the number of transit nodes or arcs in the path). The essence of the algorithm is as follows: starting from some node i, it is necessary to find an SP tree to all other nodes. Since paths of rank 1 are already found, paths of rank 2 are found during the first iteration of the algorithm. At each subsequent iteration, paths are found whose rank is larger by one than the previous rank. An iteration includes two nested loops through all nodes in the network and, hence, the asymptotic complexity of the iteration amounts to $O(n^2)$. For finding an SP tree from the node i, it is required $O((q_i - 1)n^2)$ actions, where q_i is the maximum rank of a found SP from the node i. The complexity of finding SP trees from all nodes is of the order of $O((q_{\max} - 1)n^3)$ under the assumption that we have $q_i = q_{\max}$ for all nodes.

It should be especially noted that, for finding paths with the use of a two-step criterion, it is easy to modify any well-known algorithm of finding SPs with the use of one criterion, for example, the Floyd [8] or Dijkstra [9] algorithms. It is well known that the asymptotic complexity of finding SPs from all nodes by these algorithms is of the order of $O(n^3)$.

As a rule, in solving the majority of practical problems of finding SPs with the use of two criteria, of particular interest are algorithms using a minimum of transit nodes or arcs in a path (a path rank) and a minimum path length in the capacity of optimization criteria. Therefore, this work recommends an improved version of the two-criterion algorithm (proposed in [10]) for finding all SPs in a network that satisfy the mentioned criteria. Since this article considers the finding of SPs from all nodes in a network, the complexity of the proposed algorithm will be compared with the complexity of the Floyd algorithm changed according to the same criteria. Worthy of mention is the fact that these algorithms can also be easily generalized to the case of the presence of a larger number of ordered optimization criteria.

PROBLEM STATEMENT AND SOLUTION ALGORITHM

Let a simple undirected network $G(N, P)$ with a set of nodes N, $n = |N|$, and a set of arcs P, $p = |P|$, be given. Some number $r_{kl} > 0$ is associated with each arc $p_{kl} \in P$ and is conditionally called its length. We call the number of arcs forming some path the path rank. It is required to find SPs between all nodes of the network that satisfy the criteria of minimum path rank and minimum path length if ranks are equal.

The idea of the proposed algorithm is similar to the idea of the Bellman–Shimbel algorithm [11, 12] according to which, at each conditional lth iteration, all SPs of rank up to 2^l, inclusively, are found in a network. In this case, at each subsequent iteration, all paths found during previous iterations are used and an efficient technique of representation of abstract data types (ADTs) is applied.

Let $R = ||\, r_{ij} \,||_{n\times n}$ be a topological matrix, let r_{ij} be the length of an arc p_{ij} (if such an arc p_{ij} is absent, then $r_{ij} = \infty$), let $D = ||\, d_{ij} \,||_{n\times n}$ be the matrix of lengths of the found paths, let $C = ||\, c_{ij} \,||_{n\times n}$ be a reference matrix of the found paths in which each element c_{ij}, $i \neq j$, determines the number of the penultimate node on the shortest path from i to j, $c_{ii} = 0$, $i = \overline{1, n}$, and let $Q = ||\, q_{ij} \,||_{n\times n}$ be the matrix of ranks of the found paths. Initially, D coincides with R, and, for paths of rank 1, the corresponding matrix elements $q_{ij} = 1$ and $c_{ij} = i$. For the paths that are not found, we have $q_{ij} = \infty$ and $c_{ij} = 0$. The signs $\leftarrow$, $\wedge$, and $\vee$ denote the operations of assignment, conjunction (logical "and"), and disjunction (logical "or"), respectively. The sign *** is used as a separator in comments.

Let us consider an algorithm for finding SPs with the use of a two-step criterion [10].

Algorithm SP1

1. $RMAX \leftarrow 1$; $l \leftarrow 0$.
2. $KU \leftarrow 0$; $l \leftarrow l + 1$.
3. For $\{i \mid i = \overline{1, n}\}$, execute items 4–16.
4. $KS \leftarrow 0$.
5. For $\{j \mid j = \overline{1, n}\}$, execute items 6–14.
6. If $(i \neq j) \wedge c_{ij} = 0$, then go to item 7; otherwise, execute $KS \leftarrow KS + 1$ and go to item 14.

7. For $\{k \mid k = \overline{1, n}\}$, execute items 8–12.
8. If $Q(i,k) \le RMAX \wedge Q(k,j) \le RMAX$, then go to item 9; otherwise, go to item 12.
9. $RT \leftarrow Q(i,k) + Q(k,j)$.
10. If $RT < Q(i,j) \vee (RT = Q(i,j) \wedge D(i,j) > D(i,k) + D(k,j))$, then go to item 11; otherwise, go to item 12.
11. $D(i,j) \leftarrow D(i,k) + D(k,j)$; $Q(i,j) \leftarrow RT$; $C(i,j) \leftarrow C(k,j)$.
12. Go to item 7. *** End of the loop with respect to k
13. If $C(i,j) \ne 0$, then $KS \leftarrow KS + 1$.
14. Go to item 5. *** End of the loop with respect to j
15. If $KS = n$, then $KU \leftarrow KU + 1$.
16. Go to item 3. *** End of the loop with respect to i
17. $RMAX \leftarrow 2RMAX$.
18. If $RMAX \le (n-1) \wedge KU < n$, then go to item 2; otherwise, go to item 19.
19. If $RMAX > (n-1) \wedge KU < n$, then output a message about the disconnectedness of the network.
20. End.

Let us consider a modified Floyd algorithm for finding SPs with the use of a two-step criterion.

Algorithm Floyd

1. For $\{i \mid i = \overline{1, n}\}$, execute items 2–9.
2. For $\{j \mid j = \overline{1, n}\}$, if $i \ne j$, then execute items 3–8.
3. For $\{k \mid k = \overline{1, n}\}$, if $k \ne j \ne i$, then execute items 4–7.
4. If $Q(j,i) < \infty \wedge Q(i,k) < \infty$, then go to item 5; otherwise, go to item 7.
5. If $(Q(j,k) > Q(j,i) + Q(i,k)) \vee (Q(j,k) = Q(j,i) + Q(i,k) \wedge D(j,k) > D(j,i) + D(i,k))$, then go to item 6; otherwise, go to item 7.
6. $Q(j,k) \leftarrow Q(j,i) + Q(i,k)$; $D(j,k) \leftarrow D(j,i) + D(i,k)$; $C(j,k) \leftarrow C(i,k)$.
7. Go to item 3.
8. Go to item 2.
9. Go to item 1.
10. End.

In the algorithm SP1, the loop representing conditional iterations begins with line 2; the total number of these iterations required for finding all paths in a network is accumulated in l. A conditional iteration contains three nested loops over the indices i, j, and k.

Assume that each number l, $l = 0, 1, 2, \ldots$, of a conditional iteration of the algorithm SP1 is in one-to-one correspondence with the set $\{1 + 2^l, \ldots, 2^{l+1}\}$ of path ranks. We call this correspondence the table of ranks. The correctness of the algorithm follows from the statements presented below and proved in [10].

LEMMA. If the lth iteration of the algorithm SP1 allows for finding paths with a rank larger than 2^l, then shortest paths will be incorrectly found.

Proof. In the inner loop with respect to k in the 8th line of the algorithm SP1, it is possible that $Q(i,k) < \infty$ and $Q(k,j) < \infty$. Let, at the lth iteration, paths (i,k_1) and (i,k_2) be found and, at the same time, let $2^l < Q(i,k_2) = Q(i,k_1) + Q(k_1,k_2) < \infty$ and let the number k_2 be larger than the number k_1. Since the loop with respect to k exhaustively searches for all numbers of nodes in increasing order, a node will be found whose number $k_3 > k_2 > k_1$ and such that

$$Q(i,k_2) = Q(i,k_3) + Q(k_3,k_2), \tag{1}$$

$$D(i,k_2) = D(i,k_3) + D(k_3,k_2) < D(i,k_2) = D(i,k_1) + D(k_1,k_2).$$

However, to the moment of determination of a path to k_2, we have $Q(i,k_3) = \infty$ or $Q(k_3,k_2) = \infty$, i.e., the path (i,k_3) or (k_3,k_2) is not found yet. At the lth iteration, the path (i,k_2) was marked by $C(i,k_2) = k_1$, and it will not be considered at subsequent iterations; by virtue of the fulfillment of conditions (1), this path is incorrect.

THEOREM 1. All shortest paths in a network that correspond to the table of ranks are found at each conditional iteration of the algorithm SP1.

Proof. Since, at each *l*th iteration of the algorithm, it is impossible to find paths with a rank larger than 2^l (the lemma), to find paths at this iteration, we use only the paths that have been found at the $(l-1)$th iteration. In fact, this impossibility is guaranteed by the use of the following comparison operations in the loop with respect to *k* in the 8th line: $Q(i,k) \le RMAX$, $Q(k,j) \le RMAX$, and $RMAX = 2^l$, $l = 0, 1, \ldots$, and the number $l = 0$ corresponds to the first conditional iteration. It is obvious that, at each *l*th iteration of the algorithm, all paths in a network will be found whose ranks strictly correspond to the table of ranks. Let us prove that the found paths are shortest. Let, at the *m*th iteration, a path (i, j) be obtained that does not satisfy the following criterion: a minimum rank and a minimum length. Hence, there is another shortest path that passes through a node *u* and, at the same time, $Q(i,u)$ or $Q(u,j)$ are not found. Since the *m*th iteration uses all paths found at the $(m-1)$th iteration, the inner loop with respect to *k* exhaustively searches for the paths between *i* and *j* among all the paths whose set of ranks is $\{1+2^m, \ldots, 2^{m+1}\}$ with the use of the following two-step criterion:

$$t = \arg\min \{Q(i,k) + Q(k,j)\},\ k = \overline{1,n},\ k \ne u; \qquad D(i,j) = \min \{D(i,v) + D(v,j)\},\ v \in t,$$

where *t* is the set of nodes belonging to paths having the same rank. Therefore, the path (i, j) through the node *u* could be found only at subsequent iterations and its rank would be larger, contrary to the assumption on the existence of another shortest path. The theorem is proved.

COROLLARY. To find an optimal path of rank *m* ($m \ge 2$), it is required $\lceil \log_2 m \rceil$ conditional iterations of the algorithm SP1, where the sign $\lceil \cdot \rceil$ signifies the rounding of a number up to the larger integer.

Let $S = \lceil \log_2 m \rceil$, where *m* is the maximum path rank among ranks of all shortest paths found in a network.

THEOREM 2. The time of finding all shortest paths in a network by the algorithm SP1 increases in proportion to the function $Sn^3 - n^2(3S + c) + 2n(S + c)$, where c is some constant.

Let us estimate the complexity of the algorithm SP1. We denote by $h_{k,i}$ the number of found paths from a node *i* that have a rank *k*. Then, to find all shortest paths in a network, the number of executions of the inner loop with respect to *k* by the algorithm is determined as follows:

$$\sum_{i=1}^{n} n - (1 + h_{1,i}) + \sum_{i=1}^{n} (n - (1 + h_{1,i} + h_{2,i})) + \ldots + \sum_{i=1}^{n} (n - (1 + h_{1,i} + h_{2,i} + h_{3,i} + \ldots + h_{2^{(S-1)},i}))$$

$$= \sum_{i=1}^{n} (n - (1 + h_{1,i}) + n - (1 + h_{1,i} + h_{2,i}) + \ldots + n - (1 + h_{1,i} + h_{2,i} + h_{3,i} + \ldots + h_{2^{(S-1)},i}))$$

$$= \sum_{i=1}^{n} (nS - (S + h_{1,i}S + h_{2,i}(S-1) + h_{3,i}(S-2) + h_{4,i}(S-2) + \ldots + h_{2^{(S-1)},i}))$$

$$= \sum_{i=1}^{n} \left[(nS - S) - \sum_{l=0}^{S-1} (S-l) \sum_{k \in \{\lfloor 1 + 2^{(l-1)} \rfloor, \ldots, 2^l\}} h_{k,i} \right] = Sn(n-1) - \sum_{i=1}^{n} \sum_{l=0}^{S-1} (S-l) \sum_{k \in \{\lfloor 1 + 2^{(l-1)} \rfloor, \ldots, 2^l\}} h_{k,i},$$

where the sign $\lfloor \cdot \rfloor$ signifies the rounding of a number up to the smaller integer.

The number of executions of the loop with respect to *k* depends on the structure of the initial network, and, with increasing the number of conditional iterations of the algorithm with respect to *l*, tends to $(n-2)$. Therefore, we give the following worst-case upper estimate of the algorithm:

$$T = Sn(n-1)(n-2) - (n-2) \sum_{i=1}^{n} \sum_{l=0}^{S-1} (S-l) \sum_{k \in \{\lfloor 1 + 2^{(l-1)} \rfloor, \ldots, 2^l\}} h_{k,i}. \qquad (2)$$

We denote $\sum_{l=0}^{S-1} (S-l) \sum_{k \in \{\lfloor 1 + 2^{(l-1)} \rfloor, \ldots, 2^l\}} h_{k,i} = c$ under the assumption that $h_{k,i} = h_k$ for all $i \in N$, and then rewrite estimate (2) in the form

$$T = Sn(n-1)(n-2) - (n-2) \sum_{i=1}^{n} c = Sn^3 - n^2(3S + c) + 2n(S + c). \qquad (3)$$

The theorem is proved.

Since the asymptotic complexity of the Floyd algorithm is always $O(n^3)$, it is believed that the speed of the algorithm SP1 will increase with increasing the graph or network density in contrast to the speed of the Floyd algorithm.

As is easily seen from the analysis of the algorithm SP1, its complexity can be decreased owing to decreasing the exhaustive search for nodes j in the loop in the 5th line in choosing nodes for which incoming paths are not found yet and nodes k in the loop in the 7th line in choosing nodes whose incoming paths are already found. We introduce the following data structures. Let $A = || a_{ij} ||_{n\times(n-1)}$ be a matrix containing the numbers of the nodes whose incoming paths have been found. In this case, the first matrix row contains the numbers of nodes for which the incoming paths from the first node are found, the second row contains such data concerning the second node, etc. We also introduce a vector $T = || t_i ||_n$ in which the number of nodes is accumulated for which paths from nodes i, $i = \overline{1, n}$, are found and also a vector $SP = || sp_i ||_n$ containing pointers sp_i at nodes i, $i = \overline{1, n}$, to linear one-sided lists consisting of elements of E. Each element of E consists of two fields F and AJ. The field AJ contains the number of a node whose incoming path is not found, and the field F contains the reference to the next element. The last element in the list and an empty list have null references.

Let us consider an improved version of the algorithm SP1 for finding SPs with the use of a two-step criterion.

Algorithm SP2

1. $T \leftarrow 0$.
2. For $\{i \mid i = \overline{1, n}\}$, execute items 3–12.
3. For $\{j \mid j = \overline{1, n}\}$, if $j \neq i$, then execute items 4–11.
4. If $D(i, j) < \infty$, then go to item 5; otherwise, go to item 6.
5. $T(i) \leftarrow T(i) + 1$; $A(i, T(i)) \leftarrow j$; go to item 11.
6. Form a new element $E(P)$.
7. If $SP(i).F \neq null$, then execute item 8; otherwise, execute item 9.
8. $P2 \Rightarrow P1$; $P1 \Rightarrow P$; $P2.F \Rightarrow P$; go to item 10.
9. $SP(i).F \Rightarrow P$; $P1 \Rightarrow P$; go to item 10.
10. $P.AJ \leftarrow j$.
11. Go to item 3. *** End of the loop with respect to j
12. Go to item 2. *** End of the loop with respect to i
13. $RMAX \leftarrow 1$; $l \leftarrow 0$.
14. $KU \leftarrow 0$; $l \leftarrow l + 1$.
15. For $\{i \mid i = \overline{1, n}\}$, execute items 16–31.
16. If $SP(i).F \neq null$, then go to item 17 and, otherwise, $KU \leftarrow KU + 1$; go to item 31.
17. $P \Rightarrow SP(i).F$; $P1 \Rightarrow SP(i)$.
18. Until $P.F \neq null$, execute items 19–30.
19. $j \leftarrow P.AJ$.
20. For $\{m \mid m = \overline{1, T(i)}\}$, execute items 21–26.
21. $k \leftarrow A(i, m)$.
22. If $Q(i, k) \le RMAX \wedge Q(k, j) \le RMAX$, then go to item 23; otherwise, go to item 26.
23. $RT \leftarrow Q(i, k) + Q(k, j)$.
24. If $RT < Q(i, j) \vee (RT = Q(i, j) \wedge D(i, j) > D(i, k) + D(k, j))$, then go to item 25 and, otherwise, go to item 26.
25. $D(i, j) \leftarrow D(i, k) + D(k, j)$; $Q(i, j) \leftarrow RT$; $C(i, j) \leftarrow C(k, j)$.
26. Go to item 20. *** End of the loop with respect to m
27. If $C(i, j) \neq 0$, then go to item 28 and, otherwise,
    go to item 29.
28. $T(i) \leftarrow T(i) + 1$; $A(i, T(i)) \leftarrow j$; $P1.F \Rightarrow P.F$; $P3 \Rightarrow P$; $P \Rightarrow P.F$; eliminate the element $E(P3)$. Go to item 30.
29. $P1 \Rightarrow P$; $P \Rightarrow P.F$. Go to item 30.
30. Go to item 18. *** End of the loop with respect to j
31. Go to item 15. *** End of the loop with respect to i
32. $RMAX \leftarrow 2RMAX$.
33. If $RMAX \le (n-1) \wedge KU < n$, then go to item 14 and, otherwise, go to item 34.
34. If $RMAX > (n-1) \wedge KU < n$, then output a message about the disconnectedness of the network.
35. End.

In lines 1–12, initial values for T, A, and SP are formed, and lines 13–35 form the main body of the algorithm. The vector SP it is used in outer loops with respect to i and j to choose the nodes j whose incoming paths are not found yet (lines 15–31 and 18–30). The matrix A and vector T are used in the inner loop with respect to m to choose the number of nodes k whose incoming paths from i to k are already found (lines 20–26). If an SP from i to j through some node k is found at an iteration, then j is introduced into the corresponding ith row of A, and the jth node is eliminated from the list sp_i. After the completion of the operation of the algorithm, lists sp_i are empty, and rows of the matrix A are completely filled (in the case when the corresponding network is connected).

The operation "Form a new element $E(P)$" in item 6 creates an element E and sets the pointer P to it, and the operation "Eliminate an element $E(P3)$" in item 28 deletes the element E to which the pointer $P3$ is set. Operations with the sign $\Rightarrow$ set the corresponding pointer in the left side of an expression to an element in the right side of the expression. Expressions with points of the form $SP(i).F$, $P.F$, and $P.AJ$ denotes the corresponding fields of elements to which the pointers $SP(i)$ and P are set. If it will turn out that $RMAX > (n-1)$ in line 33, then the initial network is disconnected when $KU < n$.

For the algorithms SP1, Floyd, and SP2, some modified algorithms SP1S, FloydS, and SP2S were developed for the case when the matrix R is symmetric since the property of symmetry allows one to considerably reduce the time spent for finding SPs.

The changed lines for the algorithm SP1S are as follows:

3. For $\{i \mid i = \overline{1, n-1}\}$, execute items 4–16.

5. For $\{j \mid j = \overline{i+1, n}\}$, execute items 6–14.

6. If $c_{ij} = 0$, then go to item 7; otherwise, execute $KS \leftarrow KS + 1$ and go to item 14.

11. $D(i,j) \leftarrow D(i,k) + D(k,j)$; $D(j,i) \leftarrow D(i,j)$; $Q(i,j) \leftarrow RT$; $Q(j,i) \leftarrow RT$; $C(i,j) \leftarrow C(k,j)$; $C(j,i) \leftarrow C(k,i)$.

15. If $KS = n - i$, then $KU \leftarrow KU + 1$.

18. If $RMAX \le (n-1) \wedge KU < (n-1)$, then go to item 2 and, otherwise, go to item 19.

19. If $RMAX > (n-1) \wedge KU < (n-1)$, then output a message about the disconnectedness of the network.

The changed lines for the algorithm FloydS are as follows:

2. For $\{j \mid j = \overline{1, n-1}\}$, if $i \ne j$, then execute items 3–8.

3. For $\{k \mid k = \overline{j+1, n}\}$, if $k \ne j \ne i$, then execute items 4–7.

6. $Q(j,k) \leftarrow Q(j,i) + Q(i,k)$; $Q(k,j) \leftarrow Q(j,k)$; $D(j,k) \leftarrow D(j,i) + D(i,k)$; $D(k,j) \leftarrow D(j,k)$; $C(j,k) \leftarrow C(i,k)$; $C(k,j) \leftarrow C(i,j)$.

The changed lines for the algorithm SP2S are as follows:

4. If $D(i,j) < \infty$, then go to item 5 and, otherwise, go to item 5a.

5a. If $i < j$, then go to item 6 and, otherwise, go to item 11.

15. For $\{i \mid i = \overline{1, n-1}\}$, execute items 16–31.

25. $D(i,j) \leftarrow D(i,k) + D(k,j)$; $D(j,i) \leftarrow D(i,j)$; $Q(i,j) \leftarrow RT$; $Q(j,i) \leftarrow RT$; $C(i,j) \leftarrow C(k,j)$; $C(j,i) \leftarrow C(k,i)$.

28. $T(i) \leftarrow T(i) + 1$; $T(j) \leftarrow T(j) + 1$; $A(i, T(i)) \leftarrow j$; $A(j, T(j)) \leftarrow i$; $P1.F \Rightarrow P.F$; $P3 \Rightarrow P$; $P \Rightarrow P.F$; eliminate the element $E(P3)$. Go to item 30.

33. If $RMAX \le (n-1) \wedge KU < (n-1)$, then go to item 14 and, otherwise, go to item 34.

34. If $RMAX > (n-1) \wedge KU < (n-1)$, then output a message about the disconnectedness of the network.

EXPERIMENTAL INVESTIGATION OF THE ALGORITHMS

To perform an experiment, a test program was written in the language Digital Visual Fortran (DVF) of the Digital Equipment Corporation in the environment of Microsoft (MS) Developer Visual Studio (VS) DVF 6.1. An external program interactively entered the following data: n is the number of nodes in a network; val is the number of outgoing arcs of each node (the degree of the node); boundaries of arc weight values varied from $MINVAL$ to $MAXVAL$; a parameter controlling the input–output data exchange. Hereafter, with the help of a random number generator (the built-in function $RAND(\,)$), arc

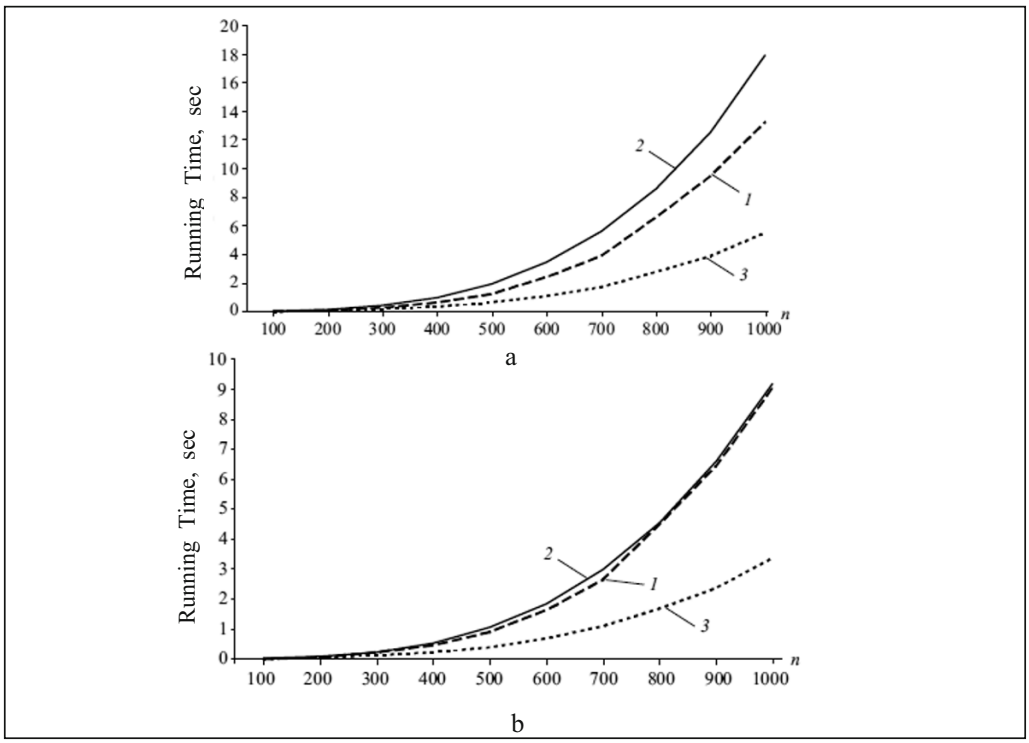

Fig. 1. Diagram of changing the running times of the algorithms SP1 (*1*), Floyd (*2*), and SP2 (*3*) (a) and SP1S (*1*), FloydS (*2*), and SP2S (*3*) (b) against the degree of nodes.

weights from *MINVAL* to *MAXVAL* were generated, and arrays R, D, Q, and C and also arrays A, T, and SP (for the algorithms SP2 and SP2S) were formed. The whole main memory necessary for the operation of the algorithms was allocated and unallocated dynamically by an external program. The running time of the algorithms was fixed by the built-in subprogram *cputime*() immediately before the start and after the completion of the algorithms of finding SPs. The work of the algorithms was tested on a PC (2.66 GHz/2 Gb) under the control of the Windows Vista operating system. The solution of the problem in the case of sparse networks with the number of nodes $n = 1000$ was independently modeled with changing the values of the parameter val from 2 to 999; with changing n from 100 to 1000 when $val = 5$; in the case of highly sparse networks, when $val = 2$ and n varied from 500 to 1000. In all these cases, it was assumed that $MINVAL = 30$ and $MAXVAL = 120$. All the algorithms were represented in the form of external subprograms with substitution of actual parameters.

Figure 1 presents the plots of the change in the running times of the algorithms SP1, Floyd, and SP2 and SP1S, FloydS, and SP2S, respectively, against the increase in the degree of nodes val from 2 to 999 for $n = 1000$.

As is easily seen from the plots in Fig. 1, the computational efficiency of the algorithms SP1 and SP1S is better than that of the algorithms Floyd and FloydS even when the degree of nodes $val = 5$, and the algorithms SP2 and SP2S considerably outrun the algorithms Floyd and FloydS with increasing the degree of nodes. For large values of the parameter val (from 200 to 999), the algorithms SP1 and SP1S turn out to be better than SP2 and SP2S. The algorithms SP2 and SP2S are always better than the Floyd algorithms and, with increasing the degree of nodes, are faster than the latter by several fold.

The plots of changing running times of the algorithms SP1, Floyd, and SP2 and SP1S, FloydS, and SP2S against the increase in the number of nodes from 100 to 1000 when $val = 5$ are presented in Fig. 2, and those for the case of increasing the number of nodes from 500 to 1000 when $val = 2$ are presented in Fig. 3. For networks with a moderate degree of sparseness, the algorithms SP1 and SP2 and also SP1S and SP2S become better than the Floyd and FloydS algorithms with increasing the number of nodes in the network. For strongly sparse networks, the algorithms Floyd and FloydS are faster than the algorithms SP1 and SP1S but are worse than the algorithms SP2 and SP2S with increasing the number of nodes.

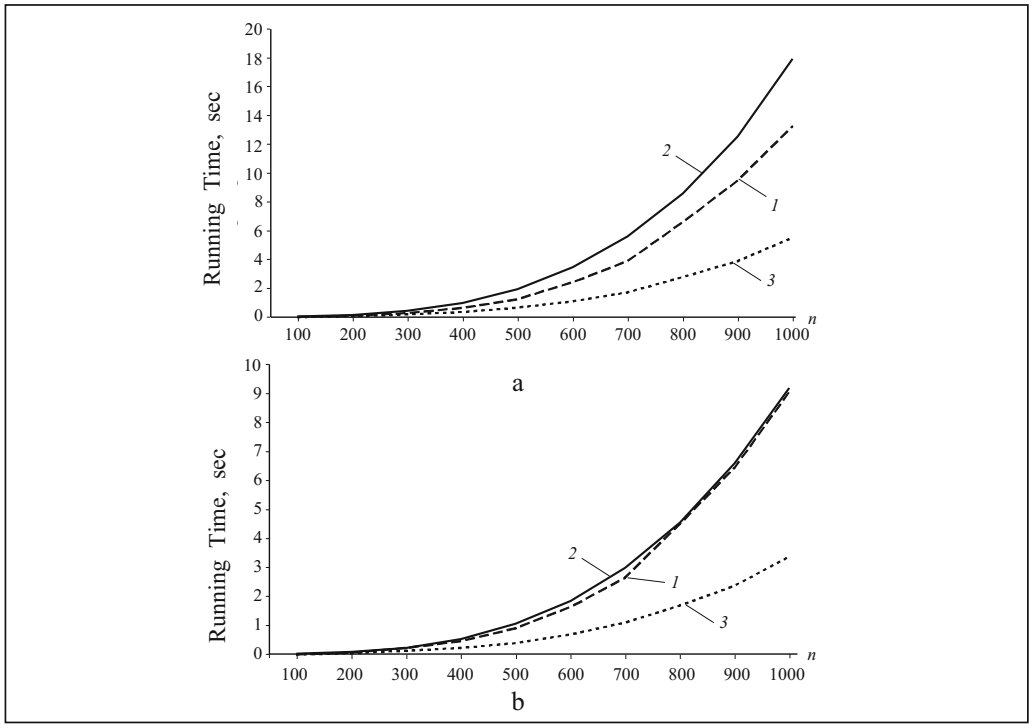

Fig. 2. Diagram of changing the running times of the algorithms SP1 (*1*), Floyd (*2*), and SP2 (*3*) (a) and SP1S (1), FloydS (*2*), and SP2S (*3*) (b) against the increase in the number of nodes when $val = 5$.

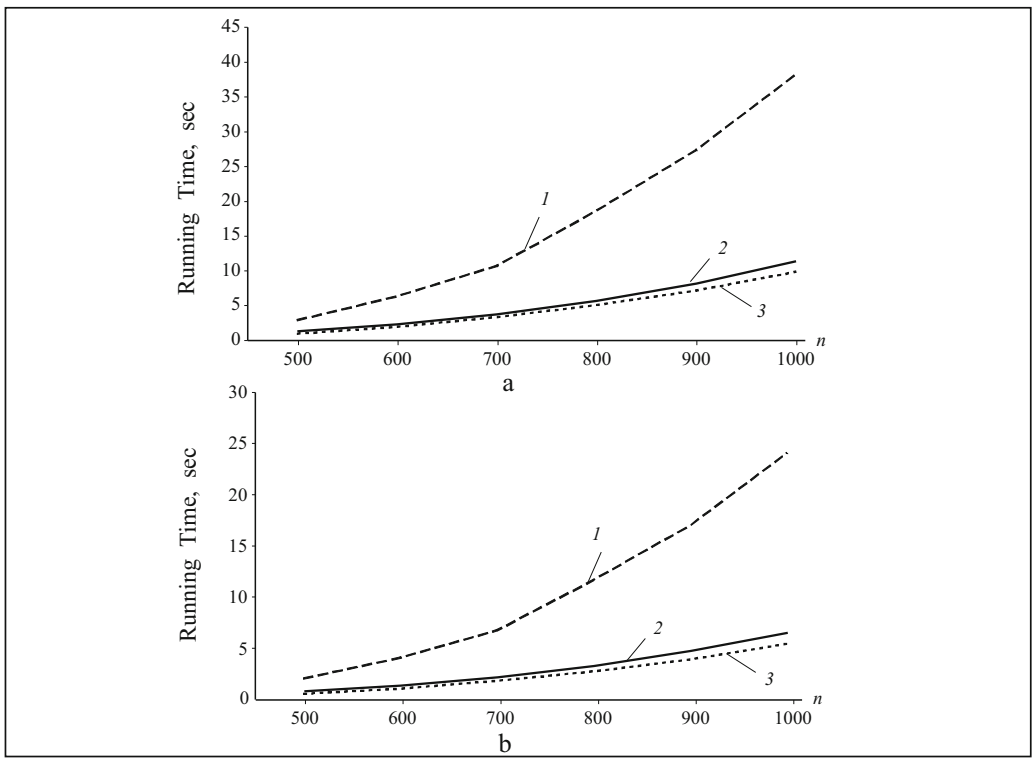

Fig. 3. Diagram of changing the running times of the algorithms SP1 (*1*), Floyd (*2*), and SP2 (*3*) (a) and SP1S (*1*), FloydS (*2*), and SP2S (*3*) (b) against the increase in the number of nodes when $val = 2$.

CONCLUSIONS

An improved version of the algorithm from [10] is considered; it finds all SPs in a network that satisfy the criteria of minimum of arcs in a path and a minimum path length. It is proved that the running time of the algorithm from [10] can be reduced owing to the use of ADTs and a decrease in the number of scanning nodes in inner loops.

It is empirically shown that the improved algorithm works faster than the algorithm from [10] on sparse networks and, with increasing the density of networks, its operating speed exceeds that of the modified Floyd algorithm by several orders of magnitude.

On the whole, the experiment has shown a high computational efficiency of the proposed algorithm that can be used as a component of a standard developer's toolkit and can be advantageously employed in solving practical problems of finding SPs with the use of two criteria on high-dimensional graphs and networks.

REFERENCES

1. P. Hansen, "Bicriterion path problems," in: G. Fandel and T. Gal (eds.), Multiple Criteria Decision Making Theory and Application, Springer, Berlin (1979), pp. 109–127.
2. A. Warburton, "Approximation of Pareto optima in multiple-objective, shortest-path problems," Oper. Res., **35**, No. 1, 70–79 (1987).
3. M. Ehrgott and X. Gandibleux, Multiple Criteria Optimization — State of the Art Annotated Bibliographic Surveys, Kluwer, Boston, MA (2002).
4. N. A. Knyazeva, "The use of logical operations for searching for optimal paths in networks," in: Proc. 2nd All-Union Conf. "Methods and programs for solving optimization problems on graphs and networks," Part 1, Theory and Algorithms, Novosibirsk (1982), pp. 85–86.
5. V. V. Dobrolyubov and V. A. Pedyash, "A network algorithm for finding paths with a minimal number of transit nodes and a maximum capacity," Computing Machinery in Engineering and Communication Systems, No. 4, 129–132 (1979).
6. E. F. Moore, "The shortest path through a maze," in: Proc. Intern. Symp. on the Theory of Switching, Part II, Harvard University Press, Cambridge, MA (1959), pp. 285–292.
7. A. V. Aho, J. Hopcroft, J. D. Ullman, The Design and Analysis of Computer Algorithms [Russian translation], Mir, Moscow (1979).
8. R. W. Floyd, "Algorithm 97: Shortest path," Comm. ACM, Vol. 5, 345 (1962).
9. F. W. Dijkstra, "A note on two problems in connection with graphs," Numerical Mathematics, Vol. 1, 269–271 (1959).
10. V. A. Vasyanin and A. I. Savenkov, "An algorithm for finding shortest paths in a network with the use of a two-step criterion," in: Diskretn. i Ergatich. Sist. Upravl, Collected Scientific Papers, Kyiv (1983), pp. 40–49.
11. R. E. Bellman, "On a routing problem," Quart. Appl. Math., **16**, No. 1, 87–90 (1958).
12. A. Shimbel, "Applications of matrix algebra to communication nets," Bulletin of Mathematical Biophysics, **13**, 165–178 (1951).