EFFICIENT ALGORITHM FOR FINDING ALL MINIMAL EDGE CUTS OF A NONORIENTED GRAPH

A. V. Karzanov and E. A. Timofeev

UDC 519.17

Consider the following problem: In an n-vertex nonoriented graph G (possibly with multiple edges), find all minimal (as to the number of edges) edge cuts. To solve this problem an algorithm is proposed with a complexity $O(\lambda n^2)$, where λ is the number of edges in a minimal edge cut of graph G. The order of growth of complexity with respect to n and λ is lower than in former algorithms. For example, the algorithm proposed in [1] has a complexity $O(nm)$, where m is the number of edges of graph G, while the complexity of the algorithm of [2] is $O(\lambda^3 n^2)$. Note that while the complexity (as to n and λ) grows in the same order as in the Ford–Fulkerstone algorithm [3], the given algorithm finds minimal edge cuts in the entire graph and not only between specified vertices as in the Ford–Fulkerstone algorithm.

The algorithm constructs a structure graph $\Gamma(G)$ (see [4]) from which any minimal edge cut of the graph G can be found in $O(n)$ operations, as follows from the following properties of the graph $\Gamma(G)$ (see [4]):

1) Any minimal edge cut of the graph G is a minimal edge cut of the graph $\Gamma(G)$ and can be found in $O(n)$ operations;

2) any two simple cycles of $\Gamma(G)$ have not more than one common vertex and the number of edges of $\Gamma(G)$ is $O(n)$.

Thus, of all presently known algorithms that find all minimal edge cuts of G, the proposed algorithm has the lowest complexity and gives all minimal edge cuts of a graph in a simple and compact form.

1. Definitions and Notation

Let V(H) be the set of vertices of graph H; (X, Y), the set of edges of the graph connecting two subsets of vertices X and Y. If $Y = \overline{X}$, $(X, \overline{X})$ is called a cut of the graph (henceforth, for the sake of brevity, an edge cut is called a cut). Further, c(X, Y) is the number of edges in the set (X, Y), and c(u, v) in particular denotes the multiplicity of edge (u, v). A cut consisting of k edges is called a k cut. The number $\lambda(G) = \min_X c(X, \overline{X})$ is called the edge connectivity of graph G. A graph G in which $\lambda(G) \geqslant k$ is called k-connected.

A k plant is a connected graph whose any two simple cycles have not more than one common vertex; the multiplicity of an edge is k if the edge does not belong to a cycle or k/2 is the edge belongs to a cycle and k is even. If k is odd the k plant has no cycles, i.e., is a tree whose every edge has a multiplicity k.

A *structural graph* of graph G is called a λ plant $\Gamma(G)$ $[\lambda = \lambda(G)]$ such that there exists a mapping $\varphi : V(G) \to V(\Gamma(G))$ having the following properties:

1) $\varphi(u) = \varphi(v)$ if and only if the vertices u and v are not separated by even one λ cut of the graph G;

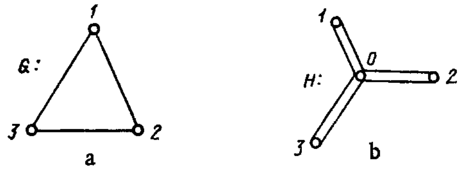
2) if $(X, \overline{X})$ is a λ cut of graph $\Gamma(G)$, then $(\varphi^{-1}(X), \varphi^{-1}(\overline{X}))$ is a λ cut of graph G and any λ cut of G can be obtained in this way.

Note that a given graph G can have several structural graphs $\Gamma(G)$. $\Gamma(G)$ then denotes any of these graphs.

For example, the graph G shown in Fig. 1 has as structural graphs both the graph G itself and the graph H, the mapping φ in the latter case not being a mapping onto V(H) (since no vertex of the graph G turns into the vertex 0).

Translated from Kibernetika, No. 2, pp. 8-12, March-April, 1986. Original article submitted March 19, 1984.

 0011-4235/86/2202-0156$12.50



a b

Fig. 1

Let f be a flow in graph G from vertex s to t; G_f is then an oriented graph (orgraph) obtained from G by replacing each edge with two oppositely oriented arcs and removing the arcs saturated by the flow f [5]; id (v) [od (v)] is the indegree (outdegree) of vertex v in the orgraph.

The vertices of the starting graph G are assumed to be numerated by the figures 1, 2,..., n. If H is a certain subgraph of G, then H^i is a graph obtained from H by contracting the set of vertices {1, 2,...,i} into one new vertex once again denoted by i.

2. Algorithm A1 for Constructing Γ(G)

In this section the algorithm A1 for constructing the structural graph Γ(G) is described, validated, and its complexity $O(nm)$ is estimated. Since the algorithm A1 is a component part of algorithm A2 whose complexity is $O(\lambda n^2)$, it is discussed in a separate section.

To simplify the description of the algorithm, let us assume that there is a certain a priori known skeleton tree G_1 of the graph G and that the numeration of the vertices of graph G is such that in the graph G_1^i the vertex i is adjacent to the vertex i + 1 (i = 1, 2,..., n – 1). [Obviously, in a connected graph this can always be carried out in $O(n^2)$ operations.]

2.1. Description of Algorithm A1

Step 0. Using the Podderyugin algorithm [1, 5] find the edge connectivity of the graph G, the number λ = λ(G).

The following transformations are next carried out in the cycle from i to n – 1 with a step 1.

Step 1. Find a flow f of power λ from vertex i to i + 1 in the graph G^i. For this purpose first find all paths consisting of one or two edges connecting i and i + 1 and then complete them to a flow of power λ using the Ford–Fulkerston algorithm [3].

Step 2. Construct the orgraph G_f^i. Find its strong-connectivity components by the algorithm described in [6]. Construct the orgraph D_f^i contracting each strong-connectivity component into one vertex.

Step 3. Numerate the vertices of orgraph D_f^i in accordance with the following rule. Let $x_1, x_2,\ldots,x_p$ be the vertices of orgraph D_f^i and $X_1^i, X_2^i,\ldots,X_p^i$, the corresponding subsets of graph G^i; then: 1) $i \in X_1^i$, $i+1 \in X_p^i$, 2) x_j is a vertex different from x_p, from which into the set $\{x_1, x_2,\ldots,x_{j-1}\}$ go at least λ/2 arcs of the orgraph D_f^i (j = 2, 3,...,p – 1).

Step 4. Record $X_1^i, X_2^i,\ldots,X_p^i$. Construct the graph G^{i+1} and turn to the next value of i.

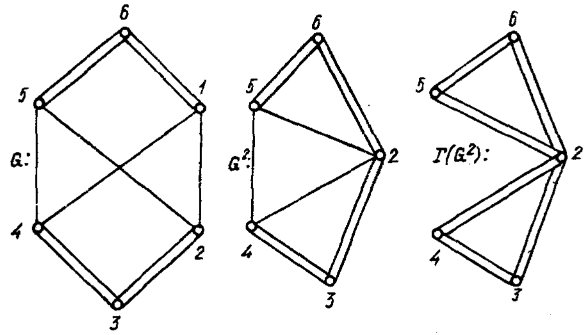
Step 5. In the cycle on i from n – 2 to 1 and with a step –1 construct a λ plant $\Gamma(G^i)$ from the λ plant $\Gamma(G^{i+1})$ and the collection of subsets of vertices of the graph G^i: $X_1^i, X_2^i,\ldots, X_p^i$. This construction takes place as follows:

1) let $V(\Gamma(G^i)) = V(\Gamma(G^{i+1})) \cup \bigcup_{i=1}^{p}\{X_i^i\} - \varphi_{i+1}(i+1)$;

2) vertices S and T of the graph $\Gamma(G^i)$ are assumed to be adjacent if and only if S and T are adjacent in $\Gamma(G^{i+1})$, or $S = X_j^i$, $T = X_k^i$ and $|i - k| = 1$, or $S = X_i^i$, $T \in V(\Gamma(G^{i+1}))$, T being adjacent to $\varphi_{i+1}(i+1)$ in $\Gamma(G^{i+1})$ and $\varphi_{i+1}^{-1}(T) \subset X_i^i$ when $\varphi_{i+1}^{-1}(T) \neq \varnothing$, or $\varphi_{i+1}^{-1}(T') \neq \varnothing$ when $\varphi_{i+1}^{-1}(T) = \varnothing$, where $T' \neq \varphi_{i+1}(i+1)$ is an adjacent to T vertex in $\Gamma(G^{i+1})$ vertex in which $\varphi_{i+1}^{-1}(T') \neq \varnothing$;

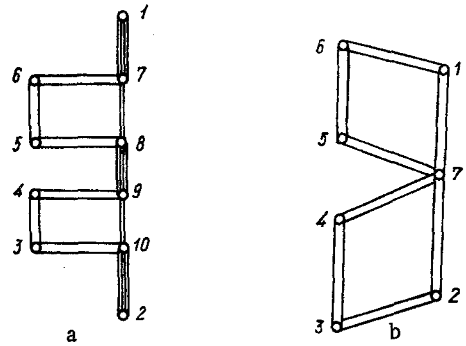
3) the mapping φ_i is defined putting

$$\varphi_i^{-1}(S) = \begin{cases} \varphi_{i+1}^{-1}(S), & S \in V(\Gamma(G^{i+1})); \\ X_j^i \setminus \varphi_{i+1}^{-1}(i+1), & S = X_j^i; \end{cases}$$



a b c

Fig. 2



a b

Fig. 3

4) for each vertex X_j^i (j = 1, 2,...,p) of the λ plant $\Gamma(G^i)$ such that $\varphi_i^{-1}(X_j^i) = \varnothing$, the degree X_j^i is 3, one of the edges incident to X_j^i has a multiplicity λ, and two others the multiplicity λ/2; the edge of multiplicity λ is contracted into one vertex.

This concludes the description of algorithm A1.

2.2. Example

Figure 2 shows the graphs G, G^2, and $\Gamma(G^2)$. It is easily verified that $X_1^1 = \{1\}$, $X_2^1 = \{6\}$, $X_3^1 = \{5\}$, $X_4^1 = \{4\}$, $X_5^1 = \{3\}$, $X_6^1 = \{2\}$. Figure 3a shows the graph Γ(G) before applying step 5.4 and Fig. 3b, after contracting the edges in step 5.4. Note that the cut ({1, 6, 5}, {2, 3, 4}) in the structural graph Γ(G) is shown twice.

2.3. Validation of Algorithm A1

To demonstrate the validity of algorithm A1 one must prove that all minimal cuts of graph G have been found and that the obtained λ plant is a structural graph of the graph G.

Steps 1 through 4 are applied n − 1 times and at each application give all minimal cuts separating vertices i and i + 1 in the graph G^i. This assertion follows from Lemmas 1 and 2. Since any λ cut of graph G is a λ cut separating vertices i and i + 1 in the graph G^i for a certain i (see [1, 5, Sec. 2.5]), one can conclude that the algorithm finds all λ cuts. To prove that the λ plant obtained at step 5 is a structural graph of the graph G, it is sufficient to show that step 5 produces a structural graph $\Gamma(G^i)$ of the graph G^i for any i. The last assertion is proved by simply testing the properties 1) and 2) of the definition of a structural graph.

Thus, to complete the demonstration of the algorithm validity it is necessary to prove two lemmas.

LEMMA 1. Let f be a flow of power k in graph G from vertex s to t; then for the arc (u, v) of orgraph G_f not to belong to a strong connectivity component of orgraph G_f it is necessary and sufficient that the edge (u, v) belongs to a k-cut separating the vertices s and t in graph G.

Proof. Necessity. Let us assume the opposite: There exists an edge (u, v) of graph G which does not belong to even one k-cut separating the vertices s and t in graph G and such that the vertices u and v lie in different strong connectivity components of the graph G_f.

To be definite, let us assume that f(u, v) > 0, so that f(v, u) = 0. Consider a set S consisting of vertex u and all vertices of graph Gf accessible from u. We will show that the set S has the following properties:

1) $s \in S$, since s is accessible from any vertex of G_f;

2) $t \notin S$, since in G_f there are at least two strong connectivity components;

3) $c(S, \overline{S}) > k$, since the cut $(S, \overline{S})$ separates s and t [in view of properties 1) and 2)] and contains an edge (u, v) not belonging to any k-cut separating s and t;

4) for any edge (p, q) $(p \in S,\ q \in \overline{S})$ $f(q, p) = 0$, since otherwise S would contain a vertex q and this contradicts the definition of set S.

Properties 3 and 4 lead to a contradiction: notwithstanding the definition of S, there exists an arc (a, b) $(a \in S,\ b \in \overline{S})$ such that $f(a, b) < c(a, b)$, i.e., G_f is the arc (a, b).

Sufficiency. Let the edge (u, v) of graph G belong to the k-cut $(S, \bar{S})$ that separates the vertices s and t $(u, s \in S, v, t \in \bar{S})$. Since for any edge (p, q) $(p \in S, q \in \bar{S})$ $f(p, q) = c(p, q)$ and $f(q, p) = 0$, the orgraph G_f includes no path from u to v, i.e., u and v lie in different strong connectivity components.

This proves Lemma 1.

Before formulating the next lemma, let us denote by D_f an orgraph obtained from G_f by contracting each strong connectivity component into a single vertex. The vertices of orgraph D_f (subsets of the vertices of graph G) are denoted by capital letters.

LEMMA 2. Let the vertices s and t of graph G be adjacent and let f be a maximal flow from s to t with a maximum power $\lambda = \lambda(G)$, the orgraph D then contains a single vertex S(T) for which od (S) = 0 [id (T) = 0] and for $|V(D_f)| \geqslant 3$ there exists a single vertex $W \neq T$ such that $c(W, S) \geqslant \lambda/2$.

Proof. Let S(T) be a strong connectivity component of orgraph G_f containing the vertex s(t) and let U be an arbitrary vertex of the orgraph D_f $(U \neq S, T)$; then, from the condition of flow conservation we have

$$\mathrm{od}(S) = \mathrm{id}(T) = 0, \quad \mathrm{id}(S) = \mathrm{od}(T) = \lambda, \qquad (1)$$
$$\mathrm{id}(U) = \mathrm{od}(U).$$

Since any cut of graph G has a power of at least λ and since for any edge (u, v) of graph G the orgraph G_f has only one arc (u, v) or (v, u), we have id (U) + od (U) $\geqslant \lambda$. Hence and considering (1) we have $(u \neq S, T)$:

$$\mathrm{id}(U) = \mathrm{od}(U) \geqslant \lambda/2. \qquad (2)$$

Since the orgraph D_f has no loops, by removing the vertex S we obtain an orgraph which also has no loops and consequently contains a vertex W with zero outdegree [$W \neq T$ if $|V(D_f)| \geqslant 3$]. Thus, in the orgraph D_f we have od (W) = c(W, S). From this and from (2) we get

$$\mathrm{od}(W) = c(W, S) \geqslant \lambda/2. \qquad (3)$$

The existence of vertex W is thus proved. This vertex is the only one that satisfies property (3) since the orgraph D includes the arc (T, S) (in view of the adjacency of vertices s and t in graph G).

This completes the proof of Lemma 2.

2.4. Estimation of the Complexity of Algorithm A1

All graphs used in algorithm A1 are specified by an adjacency list (see [6]). Step 0 has a complexity $O(n, m)$ [1, 5, Sec. 2.5]. The total complexity of step 1 is $O(n, m)$ [1, 5, Sec. 2.5]. The complexity of a single application of step 2 is $O(m)$ and of step 3, $O(m)$ since the orgraph D_f^i has no more than λn arcs; the complexity of steps 4 and 5 is $O(n)$ since the structural graph has not more than 4n vertices as follows from the following lemma.

LEMMA 3. Let $\Gamma(G)$ be a structural graph of the graph G constructed by the algorithm A1; then $|V(\Gamma(G))| \leqslant 4n$.

Proof. A minimal cut $(X, \bar{X})$ of graph G is called *parallel* if for any other minimal cut either $X \cap Y = \varnothing$, or $X \cap \bar{Y} = \varnothing$, or $\bar{X} \cap Y = \varnothing$, or $\bar{X} \cap \bar{Y} = \varnothing$. The set of all parallel cuts of graph G is denoted as P(G).

The truth of the lemma follows from the following inequalities:

$$|V(\Gamma(G))| \leqslant P(\Gamma(G)) + 1; \qquad (4)$$
$$|P(\Gamma(G))| \leqslant 2|P(G)|; \qquad (5)$$
$$|P(G)| \leqslant 2n - 3. \qquad (6)$$

Inequality (4) follows from the fact that parallel cuts of a λ plant $\Gamma(G)$ consist either of an edge of multiplicity λ or of two adjacent edges of a cycle, i.e., the number of parallel cuts $|P(\Gamma(G))|$ of a λ plant is equal to the number of edges.

Inequality (5) results from the fact that after edge contraction at step 5.4 to each parallel λ cut of the graph G^i separating the vertices i and i + 1 in graph G^i correspond not more than two parallel cuts of the graph $\Gamma(G^i)$.

Let us prove inequality (6). The proof is by induction on n. It is easy to verify that (6) holds for $n = 4$. Let us assume that (6) is true for all graphs with fewer than n vertices and show that it is also true for all graphs G having n vertices.

If for any parallel cut $(X, \overline{X})$, $|X| = 1$ or $|\overline{X}| = 1$, then $\Gamma(G)$ is a star with $n + 1$ vertices or a cycle with n vertices, i.e., $|P(G)| = n \leqslant 2n - 3$.

If for a certain parallel cut $(X, \overline{X})$, $|X| \geqslant 2$ and $|\overline{X}| \geqslant 2$, then contracting the set X (set $\overline{X}$) we get a graph G_X ($G_{\overline{X}}$, respectively). Since the cut $(X, \overline{X})$ is parallel, any minimal cut of the graph G is either a minimal cut of graph G_X or a minimal cut of graph $G_{\overline{X}}$. Thus, $|P(G)| = |P(G_X)| + |P(G_{\overline{X}})| - 1$. By assumption, we have $|P(G_X)| \leqslant 2|\overline{X}| - 1$, $|P(G_{\overline{X}})| \leqslant 2|X| - 1$. Consequently, $|P(G)| \leqslant 2|V(G)| - 2$.

This proves Lemma 3.

The complexity of algorithm A1 is thus $O(nm)$.

3. Algorithm 2 for Finding All Minimal Cuts of Graph G

This algorithm specifies all minimal cuts with the aid of structural graph $\Gamma(G)$ making use of algorithm A1 with somewhat modified steps 1 and 2.

It is assumed (as in algorithm A1) that we already know a certain skeleton tree G_1 of graph G and that the numeration of the vertices of graph G is such that in the graph G_1^i the vertex i is adjacent to vertex $i + 1$ ($i = 1, 2, \ldots, n - 1$).

3.1. Description of Algorithm A2

Stage 0. Preparation to Stage 1.

The following is assumed for $i = 1, 2, \ldots, n - 1$:

1) the flow f^i in graph G_1^i from vertex i to $i + 1$ is 0;

2) the orgraph G_f^i is equal to the graph G_1^i;

3) the orgraph $\tilde{G}_f^i$ is constructed from orgraph G_f^i by contracting edges not incident to i or $i + 1$.

Let $k = 1$.

Stage 1. Construction of structural graph $\Gamma(G_k)$, the mapping $\varphi_k: V(G_k) \to V(\Gamma(G_k))$, and the graph $J(G_k)$.

$\Gamma(G_k)$ is constructed with the aid of algorithm A1 by taking into account the flows f^i and orgraphs G_f^i and $\tilde{G}_f^i$ already found.

The following transformations are carried out in the cycle from i to $n - 1$ with a step 1.

Step 1.1. Check if there is a path connecting the vertices i and $i + 1$ in graph $\tilde{G}_f^i$ consisting of one or two edges. If such a path is found, go to step 1.3.

Step 1.2. Find a path connecting vertices i and $i + 1$ in orgraph G_f^i. Add this path to the flow f^i getting a new flow f^i (of power $k + 1$). Construct a new orgraph G_f^i and from it, an orgraph $\tilde{G}_f^i$ contracting all edges not incident to i or $i + 1$ in G_f^i. Next apply steps 2-4 of A1 and return to step 1.1 with the next value of i.

Step 1.3. Construct a new orgraph $\tilde{G}_f^i$ (the path found does not contain more than two edges). For this purpose carry out the following operations:

a) Convert the edges of the found path into arcs;

b) contract the remaining edges (these are the edges obtained at the preceding operation of Stage 2, see below).

In the orgraph $\tilde{G}_f^i$ contact the edges incident to vertices i and $i + 1$, obtaining the orgraph $\tilde{D}_f^i$ ($\tilde{D}_f^i$ will no longer have edges). Applying the algorithm described in [6], find the strong connectivity components of orgraph $\tilde{D}_f^i$. Obtain the orgraph $\tilde{D}_f^i$ after contracting each strong connectivity component into one vertex. Apply steps 3 and 4 of algorithm A1 and return to step 1.1 with the next value of i.

After steps 1.1-1.3 have been carried out for every i ($i = 1, 2, \ldots, n - 1$), apply step 5 of algorithm A1. The result will be the structural graph $\Gamma(G_k)$ and mapping φ_k.

Next construct the graph $J(G_k)$, letting $V(J(G_k)) = V(\Gamma(G_k))$ and assuming that two vertices S and T of graph $J(G_k)$ are adjacent if there exists an edge of graph G that does not belong to G_k and connects the subsets $\varphi_k^{-1}(S)$ and $\varphi_k^{-1}(T)$ in graph G.

Stage 2. Construction of (k + 1)-connected subgraph G_{k+} of graph G from k-connected subgraph G_k.

Step 2.1. Let $H = G_k$, $J(H) = J(G_k)$, and $\Gamma(H) = \Gamma(G_k)$.

Step 2.2. Find vertex U of degree k in the graph $\Gamma(H)$. If no such graph exists [i.e., $\Gamma(H)$ consists of a single vertex], assume $G_{k+1} = H$, increment k by one, and go to Stage 1.

Step 2.3. In graph J(H) find the edge (U, W). If no such edge exists, let $\lambda(G) = k$ and go to Stage 3.

Step 2.4. Let $\tilde{H} = H + (U, W)$. Construct the graphs $\Gamma(\tilde{G})$ and $J(\tilde{H})$, contracting the respective subsets of vertices of the path connecting U and W in the k plant $\Gamma(H)$. Let $H = \tilde{H}$ and go to step 2.2.

Stage 3. Construction of structural graph $\Gamma(G)$ from the available λ plant $\Gamma(H)$ of λ-connected subgraph H, where $\lambda = \lambda(G)$.

To the λ plant $\Gamma(H)$ successively add the edges of graph G that do not belong to H and connect different vertices of the λ plant. The resulting λ plant is the structural graph (G) of the graph G.

This completes the description of algorithm A2.

3.2. Validation of Algorithm A2

It is easily seen that at each application of step 2.2 the graph $\Gamma(H)$ is a structural graph of the graph H and two vertices of graph J(H) are adjacent if and only if there is an edge of graph G that does not belong to H and connects the corresponding subsets of vertices of graph G.

The inequality $\lambda(H) > k$ (at step 2.2) will hold when the graph $\Gamma(H)$ is contracted into one vertex by adding edges at step 2.4.

The algorithm A2 arrives at stage 3 when a subset of vertices U of graph G has been found such that all edges of the cut $(U, \overline{U})$ belong to G. By construction, $c(U, \overline{U}) = k$ in the graph H, and H is a skeleton graph of G such that $\lambda(H) \geqslant k$, so that $\lambda(G) = k$.

3.3. Estimation of the Complexity of Algorithm A2

To demonstrate that the algorithm A2 has a complexity $O(\lambda n^2)$ we have first to prove the following lemma.

LEMMA 4. The graph G_k has no more than kn edges.

Proof. The proof is by induction on k. The lemma is true for k = 1, since G_1 is a tree. Let us assume that G_k has no more than kn edges and show that G_{k+1} has no more than (k + 1)n edges. For this it is sufficient to prove that for a given k the number of applications of steps 2.2 through 2.4 is not more than n. In fact, by construction $\varphi_k^{-1}(U) \neq \varnothing$, $\varphi_k^{-1}(W) \neq \varnothing$. Consequently, at step 2.4 they are contracted if only two nonempty subsets of vertices of graph G_k, and since the graph G_k has n vertices, there are not more than n such contractions.

This proves Lemma 4.

Now let us show that the total complexity of applications of Stage 1 is $O(\lambda n^2)$.

Step 1.1 has a complexity $O(n)$; since it is applied not more than λn times its overall complexity is $O(\lambda n^2)$.

Step 1.2 has a complexity $O(\lambda n)$ since the number of edges in graph G_k is according to Lemma 4 not more than λn. The number of applications of step 1.2 is not greater than n – 2 (see [1]). Thus, its overall complexity is $O(\lambda n^2)$.

The complexity of step 1.3 is $O(m_i + n)$, where m_i is the number of arcs in the orgraph $\tilde{G}_f^i$ since the construction of D_f^i consists in contracting $O(n)$ edges [complexity $O(n)$] and in finding the strong connectivity components (complexity $O(m_i)$, see [6]). Since each arc of orgraph $\tilde{G}_f^i$ belongs to a single path of flow f^i and all flows f^i (i = 1, 2,...,n – 1) have no more than n – 2 paths of length $\geqslant 3$, all other paths have a length $\leqslant 2$ (see [1]), then

$$\sum_{i=1}^{n} m_i \leqslant (n-1)(n-2) + 2\lambda n \leqslant 3n^2.$$

Consequently, the total complexity of applications of step 1.3 for a given k is equal to $O(n^2)$. The total complexity of all applications of step 3 of algorithm A1 is, according to (4), $O(\lambda n^2)$. The total complexity of all applications of steps 4 and 5 of A1 is, in accordance with the results of Sec. 2.4, $O(\lambda n^2)$.

Thus, all applications of Stage 1 have a complexity $O(\lambda n^2)$.

The complexity of Stage 2 for a given k is $O(n^2)$, since according to Lemma 3 the graph G_k has no more than λn edges so that a single application of steps 2.2 through 2.4 has a complexity $O(n)$.

The complexity of Stage 3 is $O(n^2)$.

Consequently, the complexity of algorithm A2 is $O(\lambda n^2)$.

LITERATURE CITED

1. V. D. Podderyugin, "An algorithm for finding the edge connectivity of graphs," Vopr. Kibern., No. 2, 136 (1973).
2. E. A. Timoffev, "An algorithm for constructing minimax k-connected oriented graphs," Kibernetika, No. 2, 109 (1982).
3. L. R. Ford, Jr., and D. R. Falkerston, Flows in Networks, Princeton Univ. Press (1962).
4. E. A. Dinits, A. V. Karzanov, and M. V. Lomonosov, "On the structure of a system of minimal edge cuts of a graph," in: Investigations in Discrete Optimization [in Russian], Nauka, Moscow (1976), pp. 290-306.
5. G. M. Adel'son-Vel'skii, E. A. Dinits, and A. V. Karzanov, Flow Algorithms [in Russian], Nauka, Moscow (1976).
6. A. Aho, J. Hopcroft, and J. Ulman, Design and Analysis of Computing Algorithms [Russian translation], Mir, Moscow (1979).

THE EQUIVALENCE PROBLEM FOR REAL-TIME DETERMINISTIC PUSHDOWN AUTOMATA

V. Yu. Romanovskii

UDC 51:621.391

The decidability of the algorithmic equivalence problem was proved independently in [1, 2] for deterministic automata with pushdown memory (DPDA) under two constraints: 1) real-time computability; 2) empty-stack computability. In this article, we relax the second constraint and prove decidability of the equivalence problem for any two real-time DPDA. The general equivalence problem still remains open.

The decision algorithm is based on the alternate stacking construction proposed by Valiant [3]. We had to modify this model, which, although applicable directly to strict real-time DPDA, failed us in this particular case. It follows from our results that alternate stacking may diverge only for so-called "small configurations," i.e., configurations for which the norm is not too large. The notion of configuration norm for a deterministic PD-automaton introduced in this study characterizes the "distance" of the equivalence class of the given configuration from the initial configuration. The norm has the following properties: All equivalent configurations have the same norm (invariance) and there are infinitely many nonequivalent configurations with the same norm (finiteness). For strict real-time DPDA, the number of all configurations with a given norm is finite. Valiant's alternate stacking is modified by adding a certain transformation of small configurations into equivalent finite-length configurations. The properties of PD-automata are no longer sufficient for implementing this transformation. We have to shift to the class of nested stack automata, for which the emptiness problem is decidable [4]. Since the proof of existence of constants is ineffective, the question of complexity of this algorithm remains open.

Translated from Kibernetika, No. 2, pp. 13-23, March-April, 1986. Original article submitted October 15, 1984.

 0011-4235/86/2202-0162$12.50