Serdica

Bulgariacae mathematicae publicationes

Сердика

Българско математическо списание

A LINEAR ALGORITHM FOR PARTITIONING GRAPHS OF FIXFD GENUS

HRISTO NIKOLOV DJIDJEV

Any n-vertex graph of genus g has the property that it can be divided into components of roughly equal size by deleting only $O(\sqrt{(g+1)\,n})$ vertices. If such a partitioning can be found fast, then this separator theorem can be combined with divide-and-conquer method to give efficient algorithms for solving various problems defined on graphs. In the paper an algorithm is described, which finds an appropriate partitioning for a given n-vertex graph in $O(n)$ time.

1. Introduction. The problem of finding a partitioning of the vertices of a given graph satisfying certain requirements arises in connection with the use of so-called "divide-and-conquer" method [1] for finding efficient solutions to combinatorial problems. In this method the original problem is solved by decomposing it into two or more smaller problems, finding the solutions of the subproblems, and finally combining the subproblem solutions into a solution of the original problem. For problems defined on graphs the separator theorems specify different classes of graphs for which divide-and-conquer strategy can be applied efficiently. For more information about divide-and-conquer and separator theorems see [2, 3, 5, 6, 8].

The following separator theorem was proved in [8].

Theorem 1. *Let G be any n-vertex graph of genus g. The vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, $|A|\leq 2n/3$, $|B|\leq 2n/3$ and $|C|=0(\sqrt{(g+1)n})$.*

The planar separator theorem proved by Lipton and Tarjan [2] and its improvement by the author are the special case $g=0$ of Theorem 1.

The sets A and B from Theorem 1 define subproblems which are independent of each other, since no edge joins vertices belonging to different sets. The set C shows the relations between the subproblems in the original problem. The cost of combining the solutions of the subproblems into a solution of the original problem is proportional to the size of C. Thus Theorem 1 shows, that if $g=o(n)$ (then $|C|=o(n)$), divide-and-conquer approach will be useful for the class of graphs of genus g.

For the practical use of Theorem 1 it is not enough to know that a good partitioning exists, we must be able as well to construct such a partitioning fast. The proof of Theorem 1 could easily be transformed into a linear algorithm for partitioning the graph, if an imbedding of G on some orientable surface of genus g is given. Unfortunately all known algorithms for finding such imbeddings are far from effective — their complexity in the worst case is exponential with respect to the genus g.

SERDICA Bulgaricae mathematicae publications. Vol. 11, 1985, p. 369—387.

The algorithm described in this paper does not need for its proceeding neither the value of g nor the presentation of the imbedding of G on a surface of genus g. Furthermore for any graph with n vertices the algorithm finds in $0(n)$ time the suitable partitioning (which time does not depend on g).

2. Data structures. The effectiveness of the algorithm requires a convenient data representation.

Let $G=(V, E)$ be a graph and T be a breadth-first spanning tree of G. (For generally used graph theoretical terms see [9]). Any edge in G which does not belong to T is called a back edge, and the set of all back edges will be denoted by E'. The following symbols will be used in this paper to express the relations between vertices regarding T: if v is the parent of w denote $v \to w$ and if v is an ancestor of w denote $v \to \to w$.

In the algorithm which will be described here the next two problems will have to be solved many times:

A) If $v \in V$ and $w \in V$ determine whether $v \to \to w$.

B) If $v \in V$ find $y \in V$ such that there exist $x \in V$ and $(x, y) \in E'$ for which $v \to \to x$ and $\neg (v \to \to y)$ (the symbol $\neg$ denotes a logical negation).

If the vertices of G are numbered during a postorder search [1], then answers to the problem **A**) can be given in a constant time. Under the postorder search visiting and numbering of each vertex is preceded by visiting and numbering of all descendants of v. During the search for each vertex v is computed the minimum number v' of all descendants of v. Then the descendants of v are exactly those vertices in G whose numbers are between v' an v'', where v'' is the number of v.

From now on we shall denote the vertices of G by their number.

Definition. *A nearest ancestor* $N(v, w)$ *of two vertices* v *and* w *is a vertex* x *such that*

1) $x \to \to v$ *and* $x \to \to w$.

2) if $x' \to \to v$ *and* $x' \to \to w$ then $x' \to \to x$.

We shall make use of the next two properties of the nearest ancestors.

Lemma 1. *If* $(v, v_1) \in E'$, $(v, v_2) \in E'$ *and* $v < v_1 < v_2$, *then* $N(v, v_2) \to \to N(v, v_1)$.

Lemma 2. *If* $(v, v_1) \in E'$, $(v, v_2) \in E'$ *and* $v > v_1 > v_2$, *then* $N(v, v_2) \to \to N(v, v_1)$.

We shall give the proof of Lemma 1 only. The proof of Lemma 2 is analogous.

Proof of Lemma 1. Since the vertex $N(v, v_2)$ is an ancestor of v and v_2, then it is also an ancestor of all vertices x, satisfying $v < x < v_2$, whence $N(v, v_2)$ is an ancestor of v_1. Then $N(v, v_2)$ is a common ancestor of v and v_1, while $N(v, v_1)$ is their nearest ancestor. Thus $N(v, v_2) \to \to N(v, v_1)$.

Definition. *We shall call a given simple path a regular path (regarding* T*) it contains exactly one back edge.*

To be able to use the above properties of the nearest ancestor we compute for each $v \in V$ the numbers value-min (v) and value-max (v) defined by the expressions:

value-min $(v) = \min \{v'$: the path $(v_0 = v, v_1, \ldots, v_k = v')$ is regular with $(v_{k-1}, v_k) \in E'\}$,

value-max $(v) = \max \{v'$: the path $(v_0 = v, v_1, \ldots, v_k = v')$ is regular with $(v_{k-1}, v_k) \in E'\}$.

If G is a biconnected graph then for each vertex v in G except the root of T there exists a vertex in G which is no descendant of v and is adjacent to some vertex descendant of v. According to Lemma 1 and Lemma 2 one such vertex must be value-min (v) or value-max (v).

The following algorithm computes value-min (v) and value-max (v) for all vertices v in G in $0(n+m)$ time, where $m=|E|$.

Algorithm 1

1. Initial values. value-min $(v):=n+1$, value-max $(v):=0$ $v=1, 2, \ldots, n$.

2. General Step. Perform the General Step consecutively for $v=1, 2, \ldots, n$ and all edges (v, v_1).

a) If $(v, v_1) \in E'$ then current-min $:=$ current-max $:=v_1$;

b) If $v \to v_1$ then current-min $:=$ value-min (v_1);
current-max $:=$ value-max (v_1);

c) If current-min$<$value-min (v) then value-min $(v):=$current-min;

d) If current-max$>$value-max (v) then value-max $(v):=$current-max.

After the values of value-min $(\cdot)$ and value-max $(\cdot)$ have been computed, we form for each vertex v two lists of vertices of G: the first one list-min (v) contains all vertices $v_1<v$ such that $(v_1, v) \in E$ and the second, list-max (v) — all vertices $v_2>v$ such that $(v_2, v) \in E$. Both kinds of lists are sorted: the first according to the values of value-min $(\cdot)$, and the second — according to the values of value-max $(\cdot)$. Using the distributive sort [1] we can organize all lists in $0(n+m)$ time.

3. First version of the algorithm. Every orientable surface of genus 0 (i. e. surface homeomorphical to the sphere) has the following important property (Jordan's Curve Theorem [11]): Every closed curve upon the surface divides it into exactly two connected regions. As for the orientable surfaces of genus exceeding 0, there exist both closed curves dividing the surfaces into two connected regions, as well as curves which do not divide them. The deletion, however, of the points of any curve of the second type "diminishes" the genus of the resulting surface, more precisely — it can be imbedded on an orientable surface of smaller genus [8]. We shall make use of that fact further.

Now let $G=(V, E)$ be any n-vertex biconnected graph and c — a simple cycle in G. The removing of the vertices of c divides G into several (one or more) components, called segments, and let $K=(V_k, E_k)$ be the segment containing greatest number of vertices. Suppose $|V_k| \leq 2n/3$. Find a simple path p with endpoints two different vertices v_1 and v_2 from c and internal vertices (at least one) — from K. Such a path always exists since G is biconnected. The removal of v_1 and v_2 divides c into two paths p_1 and p_2, and p and c put together form two cycles c_1 and c_2, as shown in Fig. 1. Note that neither v_1 nor v_2 belongs to any of the paths p_1 and p_2, and both vertices belong to p. The removal of the vertices of p from K divides the segment into smaller segments (one or more), and let $K'=(V_{k'}, E_{k'})$ be the biggest of them. The following cases exist:

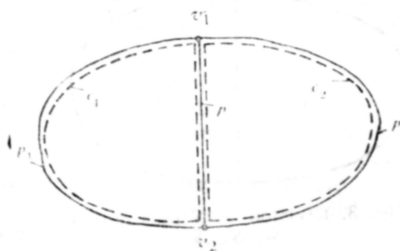

Fig. 1. The cycles c_1 and c_2

1) No edge connects a vertex in K' with a vertex in p_1 (Fig. 2 (a)). Then the removal of the vertices of c_2 divides G into segments, the biggest of which is K'. Besides $|K'|\leq|K|$, since p contains at least one vertex from K. Then the cycle c_2 partitions G "better" than c.

2) No edge connects a vertex in K' with a vertex in p_2 (Fig. 2 (b)). Then the removal of the vertices of c_1 divides G into segments, the biggest of which is K'. Then c_1 can be taken for the next iteration in the place of c.

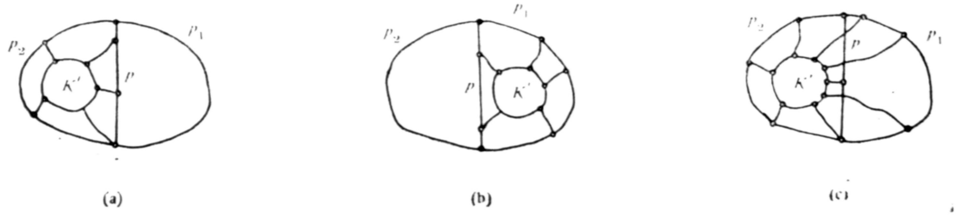

Fig. 2. The possible cases for the component K'

When either Case 1 or Case 2 applies we shall say that p divides K.

3) There exist edges connecting K' both with p_1 and p_2 (Fig. 2 (c)). Imbed G on some orientable surface S and let for any path p in G $l(p)$ denote the corresponding curve on S.

Assume that each of the closed curves $l(c_1)$ and $l(c_2)$ divides S into two parts. We shall show that this leads to a contradiction. $l(c_1)$ and $l(c_2)$ put together divide S into 3 parts S_1, S_2 and S_3 such that

$$S_1 \cap l(\breve{p})=\emptyset,\ S_2\cap l(p_2)=\emptyset,\ S_3\cap l(p_1)=\emptyset, \tag{1}$$

where $\breve{p}$ is the path containing all vertices of p except the two endpoints v_1 and v_2 (Fig. 3). Since by definition $K'\cap(c\cup p)=\emptyset$ and some edge connects K' with p, then K' belongs to one of the regions S_2 and S_3 (say S_2). In our case (Case 3) at least one edge connects a vertex in K' with a vertex in p_2 whence $S_2\cap l(p_2)\neq\emptyset$. This contradicts (1).

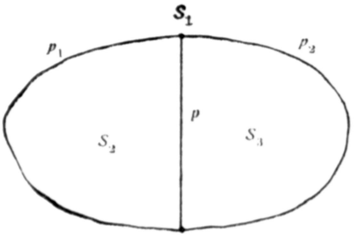

Fig. 3. Division of S into regions S_1, S_2, S_3

Therefore in Case 3) at least one of the curves $l(c_1)$ and $l(c_2)$ does not divide S into two parts. Then it follows [8] that the deletion of the vertices of both cycles c_1 and c_2 diminishes with at least one the genus of G.

Note that if Case 3 applies, then G will contain a subgraph homeomorphical to the complete bipartite graph on 2 sets of 3 vertices $K_{3,3}$ (Fig. 4).

The idea that was exhibited above leads to the following iterative procedure for partitioning G.

Algorithm 2

1. Let $G=(V, E)$ be any n-vertex biconnected graph and c_0 be a simple cycle in G. Let $K_0:=G$, $i:=0$.

* For any segment X, $|X|$ will denote the number of the vertices of X.

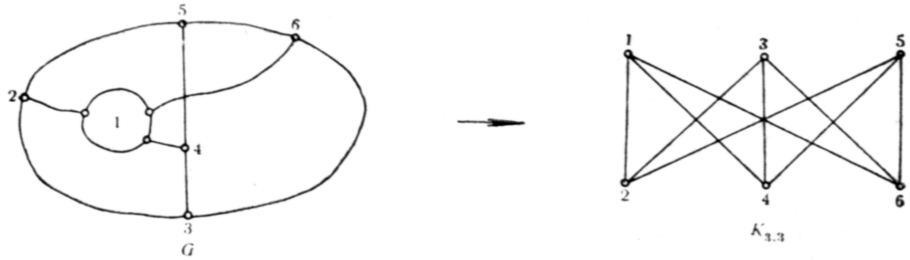

Fig. 4. The bipartite graph $K_{3,3}$

2. Find the segments into which K_i is divided by c_i. If a segment K_{i+1} with more than $2n/3$ vertices exists, then execute Step 3; otherwise execute Step 4.

3. Built a path p_{i+1} with endpoints in c_1 and the other vertices (at least one) — in K_{i+1}. Let c_i' and c_i'' be the paths into which p_{i+1} divides c_i (c_i' and c_i'' do not contain the endpoints of p_{i+1}).

If p_{i+1} divides K_i, then K_{i+1} is not adjacent with either c_i' or c_i'' (say c_i'). Let $c_{i+1}:=c_i'' \cup p_{i+1}$, $i:=i+1$ and go to Step 2.

If p_{i+1} does not divide K_i, then delete the vertices of c_i and p_{i+1} from K_i, mark the deleted vertices and denote the resulted segment by K_{i+2}. Build a new cycle c_{i+2} from vertices in K_{i+2}, increase i by 2 and execute Step 2.

4. Let C contain the vertices of $c_i \cup p_{i+1}$ plus all vertices of G marked as "deleted" in the preceding iterations. Deletion of the vertices of C partitions G into components, the biggest of which contains no more than $2n/3$ vertices. Construct sets A and B as described in the proof of Theorem 2 in [8].

Since $|K_0|>|K_1|>|K_2|\dots$ then the algorithm will halt after a finite number of steps. Let i^* be the value of the variable i at the last iteration before the algorithm terminates. Then $|K_{i^*}|>2n/3$ and p_{i^*+1} divides K_{i^*} into segments, the biggest of which contains no more than $2n/3$ vertices. Consequently C divides G into segments, the biggest of which contains no more than $2n/3$ vertices. Then it is possible to find in linear time sets A and B such that A, B, C is a partitioning of V such that no edge joins a vertex from A with a vertex from B, $|A|, |B|\leq 2\,n/3$ ([8]). From the structure of the set C, $|C|=O((g+1)\,k)$, where g is the genus of G and k is the maximum number of vertices on $p_{i+1}\cup c_i$ for $i\leq i^*$.

Throughout this paper the terms "regular parth" and "regular cycle" with regard to a certain tree are used to denote a simple path or a simple cycle with all edges except one from the tree.

Let T be a spanning tree for G with a root some vertex t and radius r. For obtaining the estimation $|C|=O(\sqrt{(g+1)n})$ from Theorem 1 it will be enough if we require that for all i each of p_{i+1} and c_i contain the vertices of no more than l regular paths (with regard to T), for some constant $l\in\mathbf{N}$. Then $k=O(r)$ and $|C|=0\,((g+1)\,r)$ which enables us to reduce size of C to $O(\sqrt{(g+1)n})$ using the idea of the proof of Theorem 2 in [8].

4. Improvements of the algorithm. To achieve the planned structure of the cycles, it will be necessary to improve the method of building the paths.

In Algorithm 2 the only restrictions put were those that the endpoints of the path p_{i+1} must be in c_i and all other vertices — in K_{i+1}. Preserving those requirements to p_{i+1}, we shall add here some new ones.

Let the path p_0 (that is the first cycle c_0) be any regular path with both endpoints the root of T. For building the next paths p_i, $i \geq 1$, we shall need the values of 3 variables B_i^1, B_i^2, and B_i^{12}. The value of each of those variables is some vertex in G and each of the vertices determines a subtree of T containing the chosen vertex and all its descendants. According to the specific situation (the details are given below) we shall require the end of p_{i+1} to belong to some of the subtrees, determined by B_i^1, B_i^2, B_i^{12}. Some relations will exist between B_i^1, B_i^2 and B_i^{12}: B_i^1 and B_i^2 are descendants of B_i^{12}, in some array E is contained the information that B_i^1 and B_i^2 form a "couple", i. e. $E(B_i^1)=B_i^2$, $E(B_i^2)=B_i^1$. It is possible that either $E(B_i^1)\neq B_i^2$, or $E(B_i^2)\neq B_i^1$, but then the values of $E(B_i^1)$ or $E(B_i^2)$ will not be used in that case.

The use of those 3 variables will give us an opportunity to give a proper "direction" to the paths; informally, the new path p_{i+1} must be a regular path (with inside vertices from K_{i+1}), the nearest common ancestor of the endpoints of which must be a vertex as near as possible to the root of the tree.

For initial values, let $B_1^1 := B_1^2 := B_1^{12} := t$ (the root of T).

For constructing the paths p_{i+1}, $i \geq 0$ use the next algorithm.

Algorithm 3. *Constructing the path*

1. Let $p_i = (x_1, x_2, \dots, x_l)$. Delete the back edge from p_i. This divides p_i into 2 paths $(x_1, x_2, \dots, x_{\bar{l}})$ and $(x_{\bar{l}+1}, x_{\bar{l}+2}, \dots, x_l)$.

2. Consider the sequence $x_2, x_3, \dots, x_{\bar{l}}$. Let $i_0 \leq \bar{l}$ be the lowest index, if any, such that

a) $\exists\ v \in K_{i+1}: (x_{i_0}, v) \in E$,

b) $\exists\ (v', w) \in E'$, $x_{i_0} \to\to v'$, $w \in K_{i+1}$,

such that

$$\neg(B_i^1 \to\to w) \text{ or } (B_i^1 \to\to B_i^2) \,\&\, \neg(x_{i_0} \to\to w) \,\&\, (B_i^1 \neq B_i^2) \tag{2}$$

(the symbols $\neg$ and & denote logical "not" and "and", respectively), and one of the following 2 conditions holds:

$$B_i^2 \to\to w, \tag{3}$$

$$\neg(B_i^{12} \to\to w). \tag{4}$$

If the conditions (2) and (3) hold then let $B_{i+1}^1 := B_i^1$, $B_{i+1}^2 := B_i^2$ and $B_{i+1}^{12} := B_i^{12}$ and if (2) and (4) hold $B_{i+1}^1 := B_i^{12}$, $B_{i+1}^2 := E(B_i^{12})$, $B_{i+1}^{12} := 0$ (the value of B_{i+1}^{12} will not be used).

If such index i_0 exists, then define p_{i+1} as the only regular path with a beginning the vertex x_{i_0} (which belongs to c_i), with an end another vertex in c_i with a back edge (v', w) (Fig. 5).

3. If such an index i_0 does not exist, then let $i^* \leq l$ be the highest index (if any) for which an edge (x_{i^*}, v) exists for some $v \in K_{i+1}$. Find a back edge (v', w) such that $x_{i^*} \to\to v'$ and $\neg(x_{i^*} \to\to w)$. Such an edge always exists, for otherwise it will follow that $x_{i^*} \to\to x$ for every $x \in K_{i+1}$, whence G will not be biconnected (x_{i^*} will be an articulation point in that case). Build a regular path p_{i+1} with a back edge (v', w) by the method of the previous step (Fig. 5). Let S_{i+1} and F_{i+1} denote the start vertex and the end of p_{i+1}, respectively. Then let $B^1_{i+1} := S_{i+1}$; if $\neg(F_{i+1} \to\to S_{i+1})$ then $B^2_{i+1} := F_{i+1}$, $E(S_{i+1}) := F_{i+1}$, $E(F_{i+1}) := S_{i+1}$, otherwise $B^2_{i+1} := B^1_i$, $E(B^1_{i+1}) := B^2_{i+1}$; if $B^1_i \to\to w$ then $B^{12}_{i+1} := B^1_i$ otherwise if $B^{12}_{i2} \to\to w'$ then $B^{12}_{i+1} := B^{12}_i$, and if neither of the two conditions is true then $B^{12}_{i+1} := F_{i+1}$.

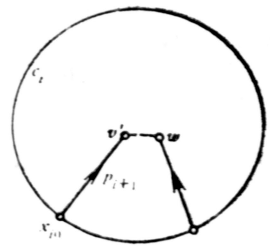

Fig. 5. Building of the path p_{i+1}
. . .Non-tree edges, →T ree edges directed from the root to the leaves; — Edges in c

4. If neither i_0 nor i^* exist, execute again Step 2 and Step 3 replacing the sequence $x_2, \ldots, x_{\overline{l}}$ by $x_{l-1}, \ldots, x_{\overline{l}+1}$ and B^1_i and B^2_i by B^2_i and B^1_i, respectively.

Algorithm 3 always constructs a path if there exists an edge connecting a vertex in p_i with a vertex in K_{i+1}. But this is always true, since

1) K_i is a connected graph (by the definition of a component).

2) K_{i+1} is set off K_i only after the deletion from K_i of the vertices of p_i.

Besides, since all paths $p_0, p_1, \ldots$ are regular, then by induction there do not exist 3 vertices x_1, x_2 and x_3 such that $x_1 \to\to x_2 \to\to x_3$, x_1 and x_3 belong to K_{i+1} and x_2 does not belong to K_{i+1}. This means that all vertices in p_{i+1}, excluding the endpoints, belong to K_{i+1}.

Now we shall prove by an induction on i that if we build the paths following Algorithm 3, then after the i-th iteration of Algorithm 2 the cycle c_i will have the structure of one of the graphs, illustrated on Fig. 6. In a concrete situation it is possible some of the parts of the cycles to be missing. For an example any regular cycle is considered of the kind (a) of Fig. 6. The simplest example of a path, the inside of which does not contain

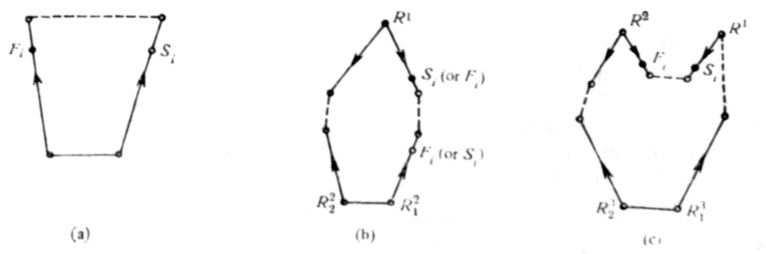

Fig. 6. The possible structures of the cycles. S_i and F_i denote the beginning and the end of p_i
. . . Path of vertices, the inside of which does not contain vertices adjacent to K_{i+1}
→ Tree edges directed from the root to the leaves
—— Path of vertices that are marked as "deleted"

vertices, adjacent to K_{i+1}, is any edge of G regarded as a path of length 1. Note, that if the inside of a path does not contain vertices adjacent to vertices in K_{i+1} then the deletion of those vertices does not influence the separation of K_{i+1} from the rest of the graph.

We shall suppose as well that a certain dependence (described below) exists between the graphs from Figure 6 and the values of B_i^1, B_i^2 and B_i^{12}.

Definition. The vertice B^1 and B^2 of G separate the branches R^1 and R^2 where R^1 and R^2 are vertices in G, if the following conditions are satisfied:

(5) $\bar{B}^1 \to\to \bar{R}^1$, $\bar{B}^2 \to\to \bar{R}^2$, $\bar{R}^1 \neq \bar{R}^2$ and either

(6) $\neg(\bar{B}^1 \to\to \bar{R}^2)$ & $\neg(\bar{B}^2 \to\to \bar{R}^1)$, or

(7) $(\bar{R}^1 \to\to \bar{R}^2)$ & $(\bar{B}^2 = \bar{R}^2)$, or

(8) $(\bar{R}^2 \to\to \bar{R}^1)$ & $(\bar{B}^1 = \bar{R}^1)$.

Let for cases (b) and (c) in Fig. 6 R^2 and R^3 denote the nearest ancestors of R_1^2 and R_2^2, and R_1^3 and R_2^3, respectively. Then the requirements that we shall put on B_i^1, B_i^2 and B_i^{12} will be the following: for the case (b) B_i^1 and B_i^2 must separate R^1 and R^2, and for case (c): B_i^1 and B_i^2 must separate R^1 and R^2, B_i^{12} and $E(B_i^{12})$ must separate the branches R^{12} and R^3, where R^{12} is the nearest ancestor of R^1 and R^2, and $B_i^{12} \to\to B_i^1$, $B_i^{12} \to\to B_i^2$.

Besides, in the case (c) for every $(x, y) \in E$ such that x and y belong to K_{i+1} and $R^1 \to\to x$, either $R^1 \to\to y$ or $R^2 \to\to y$ must be true.

Let us now suppose that after the $(i-1)$-th iteration of Algorithm 2 c_i will have the structure of some of the graphs illustrated in Fig. 6.

Suppose that p_{i+1} divides K_i. All possible cases are illustrated in Fig. 7. When analysing the structure of c_{i+1} we are taking into consideration the method in which the path p_{i+1} is built by Algorithm 3. Suppose case 3) of Fig. 7 applies (cases 1) and 2) are trivial). Since $R_i^1 \to\to F_i$ and $F_i \to\to w$, then $R_i^1 \to\to w$. Furthermore B_i^1 and B_i^2 separate R_i^1 and R_i^2 whence $B_i^1 \to\to R_i^1$ and $B_i^1 \to\to w$.

Suppose $\neg(B_i^1 \to\to B_i^2)$ holds. Then (2) is not satisfied, which shows that the path p_{i+1} has been built in Step 3 of Algorithm 3 and not in Step 2 of the same algorithm. Therefore no vertex in the path on the tree from S_{i+1} to z, excluding S_{i+1}, is adjacent to a vertex in K_{i+1} (otherwise such a vertex must be chosen in the place of S_{i+1}). Moreover it is obvious that if (x, y) is an edge such that $x, y \in K_{i+1}$ and $R_{i+1}^1 \to\to x$, then $R_{i+1}^1 \to\to y$ or $R_{i+1}^2 \to\to y$.

If $B_i^1 \to\to B_i^2$ and $B_i^1 \neq B_i^2$, then it is not possible that $R_i^2 \to\to R_i^1$ and $B_i^1 = R_i^1$ (see (8)), since $B_i^2 \to\to R_i^2$ (see (5)), whence it follows $R_i^1 \to\to R_i^2$ and $B_i^2 = R_i^2$. The condition $B_i^2 \to\to w$ ((3)) is not satisfied, because otherwise it will follow $R_i^1 \to\to R_i^2 \to\to w$ $(R_i^2 = B_i^2)$, which is not possible. Since $B_i^{12} \to\to B_i^1$ then $B_i^{12} \to\to w$ and the condition (4) is not satisfied which means that p_{i+1} has not been built in Step 3 of Algorithm 3.

The proof for the other cases from Fig. 7 is similar. Let us only pay some attention to the fact that if c_i has the structure of the graph in Fig. 6 (c), then by assumption for every edge (x, y), $x, y \in K_{i+1}$ and $R^1 \to\to x$ we have either $R^1 \to\to y$ or $R^2 \to\to y$ and consequently the cases 5), 6) and 7) of Fig. 7 are the only possible when c_i has the specified structure.

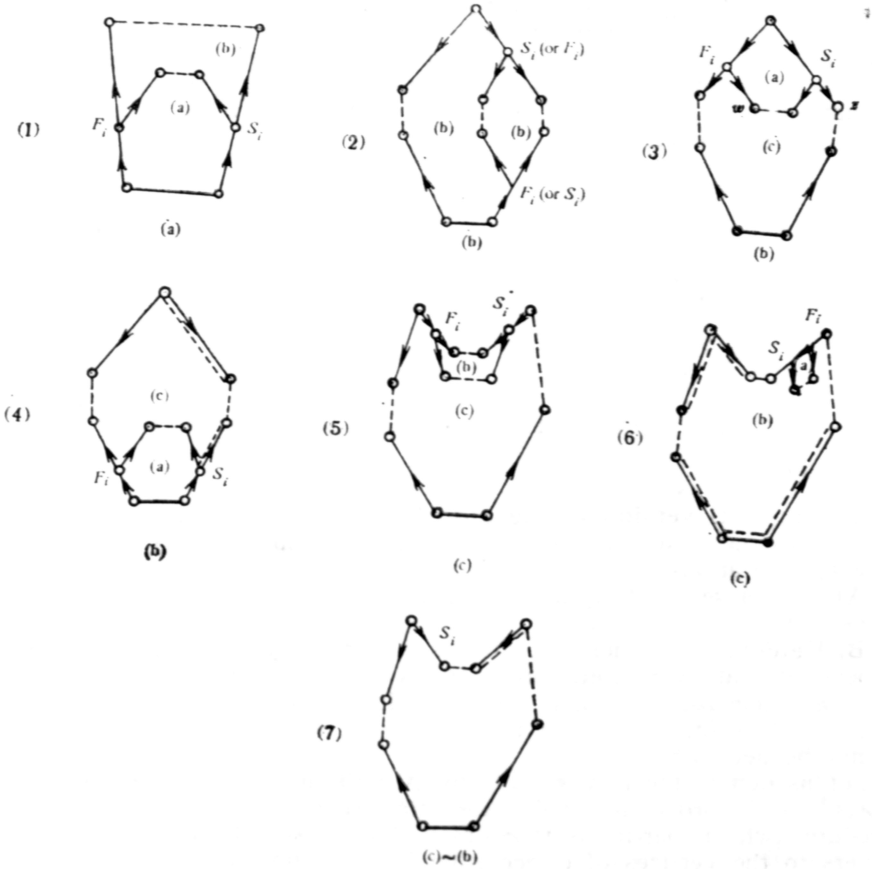

Fig. 7. The possible positions of p_{i+1}, if p_{i+1} divides K_i. For the meaning of each type line see Fig. 6. Under each of the diagrams the kind of c_i is given, and inside the diagrams — the kind of c_{i+1} according to the two possible positions of K_{i+1}. (a), (b) and (c) denote the corresponding graphs on Fig. 6

If p_{i+1} does not divide K_i, then the next operations are to be performed according to Algorithm 2: deletion of the vertices of p_{i+1} and c_i from K_i, marking those vertices as "deleted", and building the next path p_{i+2} (by Algorithm 3). Let G_i be the subgraph of G induced by the set of vertices marked "deleted" after the i-th iteration of the algorithm (i. e. the iteration before building the path p_{i+1}). By induction on i, it follows that the vertices of $G_{\overline{i}}$, adjacent to vertices of K_{i+1}, belong to the same component of $G_{\overline{i}}$. Then the cycle c_{i+1} (it contains p_{i+2} plus some vertices marked "deleted", Fig. 8), will have the structure of the graph in Fig. 6 (a).

This completes the proof that all cycles c_i have the structure of some of the graphs in Fig. 6. Therefore every cycle contains at most $5r+1$ vertices, which are not marked as "deleted", while the number of all "deleted" vertices after the execution of Algorithm 2 is not exceeding $(5r+2r)g+1$ ($2r+1$ is the maximum length of p_{i+1}). Then C contains no more than $5r+(5r+2r)g+1=(7g+5)r+1$ vertices.

Fig. 8. p_{i+2} for the case when p_{i+1} does not divide K_i. The same denotations are used as in Fig. 5—7

5. The final version of the algorithm. For the implementation of Algorithm 2 it is necessary at the i-th iteration to specify the realization of the following operations:

A) Find a segment K_i with more than $2n/3$ vertices and determine which vertices belong to it.

B) Determine whether p_{i+1} divides K_i (see Fig. 2). Solving these problems independently at every iteration requires at least $O(n)$ time for one iteration and $O(n^2)$ time for the whole execution. It is possible, however, to reorganize Algorithm 2 in such a manner that the direct solution of the above problems will not be necessary.

Let us denote the new version by Algorithm 4. The algorithm consists of 2 parts — main procedure and subroutine SEARCH. SEARCH is a recursive procedure, which partitions the graph into a set of paths $p_0, p_1, \ldots$, gives numbers to the vertices of G according to the number of the first path which contains them, counts the vertices of the subgraphs of G corresponding to the segments from Algorithm 2. The numbers, given to the vertices of G will be later used to define the partitioning A, B, C. In the new version the path p_{i+1}, $i=0, 1, \ldots$ is built in a similar way as in Algorithm 3, only the requirement that the inside of p_{i+1} must contain vertices of a segment with more than $2n/3$ vertices is removed. Another feature of SEARCH it that the cycle c_i, $i=1, 2, \ldots$ is not explicitly determined, instead the segments are defined (by the numbers of the vertices) into which c_i divides G. The sequence in which the, segments are searched is arbitrary (not necessarily the biggest segment first) until at last a segment with at least $n/3$ vertices is encountered. If the examined segment contains at least $n/3$ vertices, then during the search in that segment the desired partitioning of the graph is found. The procedure SEARCH is presented with details below.

The main procedure determines the components and the bicomponents of the graph. If the biggest component contains more than $2n/3$ vertices, then an appropriate data structure for that component is constructed (the one described in Section 2 of the paper) and through SEARCH constructs a partitioning

A', B', C' of the vertices of that component. This partitioning is finally transformed into a partitioning A, B, C of the vertices of whole graph, satisfying $|A|$, $|B| \leq 2n/3$, $|C| \leq \sqrt{21g+15}\,\sqrt{n}$.

Steps 10-15 of the main procedure are grounded on the proof of Theorem 2 in [8].

Algorithm 4.

Main procedure

1. Let G be a graph with n vertices and m edges.

If $m \geq 4n$ then let A contain any $2n/3$ vertices, let $B = \emptyset$ and let C contain the remaining vertices of G.

If $m < 4n$ go to Step 2.

2. If G is not connected, proceed as in [2]. Suppose that G is connected. Find the bicomponents of G using the algorithm in [10] with complexity $O(n+m)$. Keep information about the number of the vertices of each bicomponent.

3. Let H be a bicomponent of G such that the deletion from G of all the vertices of H except the articulation points (i. e. vertices that belong to more than one bicomponent) leads to a graph (possibly a null graph) which has no component with more than $2n/3$ vertices. Such a bicomponent can be found easily by a slight modification of the algorithm in [10] for finding the bicomponents (in Step 2 above). Denote by t an arbitrary vertex in H.

If H contains a single vertex (the vertex t), then the removal of t divides G into components, the biggest of which contains no more than $2n/3$ vertices. Then let $C = \langle t \rangle$ and let A, B be a partitioning of the vertices of the graph $G-t$ satisfying $|A|$, $|B| \leq 2n/3$.

If H contains at least 2 vertices then go to Step 4.

4. Find a breadth-first spanning tree T with a root t using the algorithm in [10] (complexity $O(n+m)$). Denote by r the radius of T, by $L(l)$ the number of the vertices on level l and by parent (v) the parent of the vertex $v \neq t$.

5. Number the vertices in postorder using the algorithm in [1]. From now on use the numbers of the vertices as their names.

6. For each $v \in V$ compute the numbers value-min (v) and value-max (v) using Algorithm 1.

7. For each $v \in V$ form the lists list-min (v) and list-max (v) defined in Section 2.

8. Find a vertex x, such that n (the root of T by the new notation) is a parent of x and the condition $\neg(x \to\to$ value-min $(x))$ is satisfied (in the next section the existence of such a vertex x is proved). Execute the procedure SEARCH. Initial values of the static variables used in SEARCH: number $(v) :=$ path $(v) := 0$, $1 \leq v \leq n$; A-start $:=$ current-number. These variables have the following meanings: number (v) is the number given to the path v during the search, path (v) is the number of the path containing the vertex v, current-number is the highest number given to any path up to the moment, and A-start is the lowest number of a path whose inside vertices belong to A.

Values of the parameters of SEARCH: start $= x$, dir-value $=$ min (start), sum $= 0$, $B1 = B2 = B12 = n$, list $=$ list-min.
As a result of the execution, the correct values of number (v), $v = 1, 2, \ldots, n$ and A-start are computed.

9. Let $A' = \{x \in V:$ number (path $(x)) \geq$ A-start$\}$,
$C' := \{x \in V \setminus A' : \exists\, (x, x_1) \in E,\ x_1 \in A'\}$, $B' := V \setminus (A' \cup C')$.

10. For each level l compute the number $L_c(l)$ of the vertices which belong both to C' and to level l.

11. For each $\alpha \in (0, 1)$ let l_α denote a level such that

$$\sum_{l=0}^{l_\alpha - 1} L(l) < \alpha n, \quad \sum_{l=0}^{l_\alpha} L(l) \geq \alpha n.$$

Compute the levels $l_{1/3}$ and $l_{2/3}$. Let l^1 be a level between $l_{1/3}$ and $l_{2/3}$ such that

$$L(l^1) = \min \{L(l) : l_{1/3} \leq l \leq l_{2/3}\}.$$

12. Find the integer j satisfying

$$\sum_{l=l_{1/3}-j+1}^{l_{2/3}+j-1} L(l) < 2/3n, \quad \sum_{l=l_{1/3}-j}^{l_{2/3}+j} L(l) \geq 2/3n.$$

Let i^*, $0 \leq i^* \leq j$ minimizes the sum $L(l_{1/3}-i) + L(l_{1/3}+i)$ for $0 \leq i \leq j$. Denote $l_1^2 = l_{1/3} - i^*$ and $l_2^2 = l_{2/3} + i^*$.

13. Find levels l_1^3 and l_2^3, minimizing the sum

$$S(l_1, l_2) = L(l_1) + \sum_{l=l_1+1}^{l_2-1} L_c(l) + L(l_2), \quad 0 \leq l_1 \leq l_{1/3} - j - 1 < l_{2/3} + j + 1 \leq l_2 \leq r+1.$$

14. If $\min \{L(l^1), L(l_1^2) + L(l_2^2)\} \leq S(l_1^3, l_2^3)$, then define the sets A, B, C as described in the proof of Theorem 2 in [8]. Else go to Step 15.

15. Denote by C the set of the vertices on levels l_1^3 and l_2^3 plus all vertices in c' which lie on levels l_1^3+1 through l_2^3-1. The deletion of the vertices of C divides G into components, the biggest of which contains no more than $2n/3$ vertices. Then the sets A and B are formed as in the case of nonconnected graphs.

End of the main procedure.

Recursive procedure SEARCH

Parameters: start, dir, sum, $B1$, $B2$, $B12$, list.

Meaning of the parameters:

start — the beginning of the next path p;

dir — a vertex in the path p (corresponds to w in Fig. 5);

sum — the number of the vertices of the segment, containing p;

$B1$, $B2$, $B12$ — correspond to B_i^1, B_i^2, B_i^{12};

list — one of the lists list-min and list-max.

A local variable: list-unnumbered $(\cdot)$. For each vertex v in the graph it provides a list (possibly empty) of previously built paths with start vertex v, to which paths no number is given up to the moment.

The variables number $(\cdot)$, path $(\cdot)$, A-start and current-number (defined in the main procedure), as well as the array $E(\cdot)$ defined in the previous section and the lists list-unnumbered $(\cdot)$, are common for all executions of SEARCH, while for the other variables new generations are created for every new call of SEARCH.

1. {Build the regular path p from start to dir.}

Initial values: $v_1 :=$ start, $p := (v_1)$, sum $:= 0$.

Do Steps 1.1 — 1.2 while $v_1 \neq$ dir.

1.1. Delete from list (v_1) its top element v_2 and add v_2 to the end of p.
1.2. $v_1 := v_2$.
2. {Build the path down the tree beginning with dir and add that path to p.}
Do Steps 2.1—2.2 while $v_1 \neq n$ (the root of the tree) and path $(v_1)=0$.
2.1. $v_1 :=$ parent (v_1).
2.2. Add v_1 to p.
3. {Storing information about the new path.}

3.1. For each vertex v from p, path $(v) := \breve{p}$, where $\breve{p}$ is the second vertex of the path (second vertices are used as names of the paths).

3.2. If number (path $(v_1)) = -1$ (i. e. if the last vertex v_1 of p belongs to an "unnumbered" path p') then number (path $(v_1)) := 0$ and include the name of the path in list-unnumbered (start′), where start′ is the first vertex of p'.

3.3. number $(\breve{p}) :=$ current-number $:=$ current-number $+1$.

Second entry for SEARCH. Will be used for paths already built during the previous iterations, the corresponding segments of which have not been searched. Parameters: $\breve{p}$, sum, $B1$, $B2$, $B12$.

4. {Updating some variables.}

If $B2=0$ then do Steps 4.1 or 4.2 and if $B12=0$ — Step 4.3 (see Step 6.1.3).

4.1. If $v_1 \rightarrow\rightarrow$start, then $B2 := v_1$, $E(B1) := B2$, $E(B2) := B1$.

4.2. Else $B2 := B12$, $E(B1) := B2$ (the array E has the same meaning as in Section 4 of this paper).

4.3. $B120 := v_1$.

5. {Searching the segments.}

Let $p=(w_0, w_1, \ldots, w_k, w_{k+1}=\text{dir}, \ldots, w_m)$, $p'=(w_0, w_1, \ldots, w_k)$.
$p''=(w_m, w_{m-1}, \ldots, w_{k+1})$.

Cycle: do Steps 5.1 and 5.2 for $i=0, 1, \ldots, k$.

5.1. Do Steps 5.1.1—5.1.4 if list-min (w_i) is a nonempty list.

5.1.1. Delete one by one the top vertices in list-min (w_i) until either the list becomes empty or the top vertex w' satisfies $\neg(w' \rightarrow w_i)$ and path $(w')=0$.

5.1.2. If $w_i \rightarrow w'$, then dir′ := value-min (w'), else dir′ $:= w'$.

5.1.3. Determine whether the condition (2) and one of the conditions (3) and (4) (defined in Section 4) are satisfied for $B_i^1=B1$, $B_i^2=B2$, $B_i^{12}=B12$, w = dir′. If the answer is "no" then go to Step 5.2: if (3) is satisfied then let $B1' := B1$, $B2' := B2$, $B12' := B12$; and else if (4) is satisfied then let $B1' := B12$, $B2' := E(B12)$.

5.1.4. If $i=0$ (then w_i is an endpoint) then perform a) and b), else perform c) — g).

a) Execute Steps 1, 2, 3.1 above with start $= w_i$ and dir = dir′. Let $\bar{p}$ be the name of the constructed path.

b) number $(\bar{p}) := -1$; add $\bar{p}$ to list-unnumbered $(\bar{w}_i)$, where $\bar{w}_i$ is the start vertex of the first path containing w_i.

c) Call SEARCH with values of the parameters w_i, dir′, sum′, $B1'$, $B2'$, $B12'$ and list-min, respectively. The result of the execution (if a solution has not been found during the search; see Step 5.1.4 e)) is a computation of the number sum′ of the vertices of the segment containing dir′.

d) sum := sum + sum′.

e) If sum $\geq n/3$, then A-start := number $(\breve{p})+1$; return back to the point of the call in the main procedure.

f) If list-unnumberd (start) is a nonempty list then for all paths $\bar{\bar{p}}$ with names in that list such that number $(\bar{\bar{p}}) \leq 0$ do the following: number $(\bar{\bar{p}})$: = number $(\breve{p})$ — 1; execute SEARCH recursively with entry at Step 4 and parameters $\bar{\bar{p}}$, sum, $B1'$, $B2'$, $B12'$; number $(\bar{p})$: = current-number: = current-number + 1.

g) If sum $\geq n/3$ then A-start: = number' $(\breve{p})+1$ and return to the main procedure.

5.2. Do Step 5.1 replacing list-min and value-min by list-max and value-max.

End of the cycle.

6. Cycle: do Steps 6.1—6.4 for $i=k, k-1, \ldots, 1$.

6.1. Do Steps 6.1.1—6.1.4 while list-min (w_i) is a nonempty list.

6.1.1. Delete one by one the top vertices in list-min (w_i) until either the list becomes empty or the top vertex w' satisfies $\neg(w' \rightarrow w_i)$

6.1.2. If $w_i \rightarrow W'$, then dir': = value — min (w'), else dir': = w'.

6.1.3. If the condition

$$\neg(w_i \rightarrow\rightarrow \mathrm{dir}') \tag{9}$$

is satisfied, then $B1'$: = w_i, $B2'$: = 0 (the right value will be assigned after the endvertex of the path is found — in Steps 4.1 and 4.2 of the next execution of SEARCH), $B12'$: = $B1$ if $B1 \rightarrow\rightarrow w_i$ or $B12'$: = $B12$ if $B12 \rightarrow\rightarrow w_i$, or $B12'$: = 0 if neither of the conditions is satisfied (see Step 4.3).

6.1.4. If the condition (9) is satisfied do Steps 5.1.4 c) — g).

6.2. Do Step 5.1 replacing list-min and value-min by list-max and value-max.

6.3. If w_i is an articulation point, then for each unnumberd descendant z of w_i do Steps 6.3.1—6.3.4.

6.3.1. Let $H^* = \{x \in V : z \rightarrow\rightarrow x\}$. The set H^* can be constructed easily by using the fact that the vertices of G are numbered in postorder (in Step 5 of the main procedure).

6.3.2. Let $\bar{v}$ be an arbitrary vertex of H^* ($\bar{v}$ will be used as a name of the next path which will contain all vertices of H^*).

a) current-number: = current-number + 1.

b) For each vertex v in H^* (including v) path (v): = $\bar{v}$.

c) number $(\bar{v})$: = current-number.

6.3.3. sum: = sum + $|H^*|$.

6.3.4. If sum $> 2n/3$ then A-start: = current-number and return to the main procedure. If $n/3 \leq$ sum $\leq 2n/3$ then do Step 5.1.4 e).

6.4. If $w_i \neq$ start and $w_i \neq v_1$, or if number $(\breve{p}) = 1$ then sum: = sum + 1.

End of the cycle.

7. Do Steps 5 and 6 replacing p' by p'', indices $0, 1, \ldots, k$ by $m, m-1; \ldots, k+1$ and exchanging the values of $B1$ and B 2.

8. Restore the values of $B1$ and $B2$ and return to the point of the call.

End of the procedure.

6. Analysis of the algorithm. We shall first prove the correctness of Algorithm 4, showing its close connection with Algorithm 2 and Algorithm 3.

If $m \geqq 4n$ (see Step 1 of the main procedure), then according to Theorem 4.2 in [12] $m \leqq 3n+6g$ whence $n \leqq 6g < 12\, g+6$. Then $|c| = \lceil n/3 \rceil \leqq n = \sqrt{n}\sqrt{n} < \sqrt{12g+6}\,\sqrt{n}$ and the partitioning A, B, C satisfies Theorem 1.

Consider now the case $m < 4n$.

Each path constructed by SEARCH has a startpoint the vertex start and all its edges except one (the edge (x, dir)) belong to T. Such a path will be regular iff it is a simple path. Let $p_o, p_1, \ldots,$ be the paths built by SEARCH, in the same order in which they are numbered by the algorithm.

Let us prove that p_o is a simple path. Since H contains at least 2 vertices (see Step 3 of the main procedure), then according to Theorem 3.3 from [9] there exists a simple cycle containing the root n of T. Let $c = (n, x_1, x_2, \ldots, x_l, n)$ be a simple cycle such that x_1 has a maximum value among all simple cycles containing the root of T. Since c is simple, then $x_1 \neq x_l$ and since both vertices are children of n, then x_l is not a descendant of x_1. Let $i > 1$ be the minimum index for which x_i is not a descendant of x_1. Then (x_{i-1}, x_i) is an edge which does not belong to T, $x_1 \rightarrow\rightarrow x_{i-1}$ and $\neg(x_1 \rightarrow\rightarrow x_i)$. Furthermore $x_i < x_1$, since the assumption $x_i > x_1$ contradicts to the extremal property of x_1. Then value-min (x_1) is a vertex which is not a descendant of x_1, whence the vertex $x =$ start from Step 8 of the main procedure exists. Hence for the vertex dir $=$ value-min (start)

(10) $$\neg(\text{start} \rightarrow\rightarrow \text{dir})$$

holds. Using the values of start and dir, in Steps 2 and 3 of SEARCH a path is constructed (the path p_0'), which begins with the vertex start, goes up the tree to a vertex x such that (x, dir) is an edge that does not belong to the tree, and ends with the path down the tree to either the first vertex included in a path or the root of the tree. The condition (10) shows that dir is not a descendant of any vertex in the path from start to x. Then the path p_0 is simple.

The condition (10) holds as well for the values of start and dir when building the paths p_i, $i \geqq 1$ (Steps 2 and 3 of SEARCH), since one of the conditions (2) and (9) has been satisfied at the previous iteration. Then the paths $p_1, p_2, \ldots$ are also regular.

Consider the iteration just betore Steps 4-7 of SEARCH are executed for the path p_{i+1}. Denote by M_i the set of the vertices of G that belong to a path built before the considered iteration. Let $G_i = G - M_i$ and let $K_i^1, K_i^2, \ldots, K_i^s$ be the components of G_i. If none of those components is adjacent to a vertex from the inside of p_i, then in Steps 5, 6 and 7 of SEARCH it is established that there does not exist a regular path with length at least 2 which starts with a vertex from the inside of p_i and the interior vertices of which belong to G_i. That means that in this case no immediate recursive call of SEARCH is to be performed. In this case the execution continues on a lower level of recursion.

Suppose now that some of the components $K_i^1, K_i^2, \ldots, K_i^s$ are adjacent to a vertex from the inside of p_i. Then the vertices from the inside of p_{i+1} belong to one of those components (say K_i^1) and let K_i^1 be not a bicomponent of G. We shall prove that either

a) during the recursive execution of SEARCH at some iteration the inequality sum $\geqq n/3$ (Step 5.1.4 e) has been satisfied and the execution continued in the main procedure, or

b) after the recursive execution all vertices of K_i^1 belong to numbered paths (we shall call such vertices numbered) and sum becomes equal to $|K_i^1|$.

Suppose the inequality sum$\geq n/3$ has not been satisfied during the recursive execution of SEARCH and let $p_{i+1}, p_{i+2}, \ldots, p_{i+i_1}$ be the numbered paths constructed during that execution. As stated above, all vertices from the inside of p_{i+1} belong to K_i^1. If the start-vertex v of p_{i+2} belongs to the inside of p_{i+1}, then all inside vertices of p_{i+2} also belong to K_i^1. Suppose v is an endpoint of p_{i+1}. Since p_{i+2} is numbered, then p_{i+2} belongs to the list-unnumbered (v) list (see Steps 3.2 and 5.1.4 f)). This list, by construction, contains those unnumbered paths with start v, the inside of which is reached during the recursive call of SEARCH. Then some vertex w from the inside of p_{i+2} is adjacent to a vertex from the inside of a path among $p_{i+3}, \ldots, p_{i+i_1}$. Assume that the insides of $p_{i+3}, \ldots, p_{i+i_1}$ belong to K_i^1. Then w is adjacent to a vertex in K_i^1, which means that w (which is from G_i) belongs to K_i^1. Hence all vertices from the inside of p_{i+2} belong to K_i^1. By induction, the insides of all paths $p_{i+2}, \ldots, p_{i+i_1}$ belong to K_i^1.

The necessity of creating list-unnumbered ($\cdot$) lists is due to the following. If p_{i+2} has a startvertex from the inside of p_{i+1} then all vertices from the inside of p_{i+2} must belong to the same component of G_i that contains the inside of p_{i+1}, i. e. K_i^1. If, however, the startvertex is an endpoint of p_{i+1}, then we do not know in advance whether the inside of p_{i+2} is from K_i^1 or not. In that case we give number to p_{i+2} only after it is reached by another path which belonging to K_i^1 is established. If p_{i+2} is not reached, then we number it after returning to the path containing v in its inside (this path must be among $p_0, p_1, \ldots, p_i$).

Now we shall prove that each vertex from K_i^1 belongs to some of the paths $p_{i+1}, p_{i+2}, \ldots, p_{i+i_1}$. Suppose that a vertex x from K_i^1 exists which belongs to neither of the paths. Since K_i^1 is connected, then a path $x=x_1, x_2, \ldots, x_l$ exists such that $x_1, x_2, \ldots, x_{l-1}$ belong to K_i^1 and x_l belongs to the inside of p_{i+1}.

Let $j>1$ be the lowest index for which x_{l-j} belongs to neither of the paths $p_{i+1}, \ldots, p_{i+i_1}$. Then the vertex x_{l-j+1} belongs to some of those paths. Let p_{i^*}, $i+1\leq i^*\leq i+i_1$ be the path that contains x_{l-j+1} in its inside. Then during the execution of Steps 5, 6, 7 of SEARCH for p_{i^*} either x_{l-j} will be numbered (in the case when x_{l-j} does not belong to any path among $p_0, p_1, \ldots, p_{i^*}$), or the unnumbered path p^* containing x_{l-j} will be included in a corresponding list-unnumbered (s^*) list (s^* is the startvertex of p^*). Since in the second case p^* can be built only after p_{i+1}, then all paths from list-unnumbered (s^*) will be numbered before the search in K_i^1 is finished and thus p^* must be some of the paths $p_{i+1}, \ldots, p_{i+i_1}$. The contradiction shows that each vertex from K_i^1 belongs to some path among $p_{i+1}, \ldots, p_{i+i_1}$.

Then during the recursive execution of SEARCH all vertices of K_i^1 and only they are numbered and sum becomes equal to $|K_i^1|$ (see Step 5.1.4 d)).

In the same way we prove that if some of the components $K_i^2, \ldots, K_i^s$ is adjacent to an inside vertex of p_i, then during the next execution of

SEARCH the vertices of exactly one of those components are numbered, and so on. If some vertex w in p_i is an articulation point, then the vertices of all bicomponents incident with that vertex (except the one containing the root) will be consequently numbered in Steps 6.3.1 and 6.3.2 of SEARCH. According to the definition of biconnectivity, the set H^* (used in Step 6.3) of the vertices of these bicomponents is exactly the set of all descendants of those children of w which belong to no path.

In result of the choice of H in Step 3 of the main procedure each of the above components contains no more than $2n/3$ vertices. If $|H^*|\geqq n/3$, then the partitioning set C' will contain only w; if $|H^*|<n/3$, then to the current value of sum is added $|H^*|$ (Step 6.3.3) and the search in the current component will be continued.

Suppose that after the search in the components $K_i^1, \dots, K_i^{s'}$, $s'\leqq s$, the inequality sum$\geqq n/3$ holds (s' is the minimum index with that property). Since each of the components $K_i^1, \dots, K_i^{s'}$ contains less than $n/3$ vertices (otherwise the condition sum$\geqq n/3$ would have been satisfled earlier and the execution — continued in the main procedure), $n/3\leqq \Sigma_{j=1}^{s'}|K_i^j|\leqq 2n/3$ holds. Then if A' is the set of vertices in $K_i^1, \dots, K_i^{s'}$, C' — the set of vertices not belonging to A' and adjacent to vertices in A', and B' — the set of vertices which remain (these sets are built in Step 9 of the main procedure), then $|A'|$, $|B'|\leqq 2n/3$.

In order to prove that $|C'|\leqq(7g+5)r+1$ we show that if G is biconnected, then the same set A' can also be constructed as a result of applying Algorithm 2 to G. Define a tree Q showing the relation between the recursive calls of SEARCH. Each vertex in Q corresponds to a path constructed by SEARCH and p_0 is the root of Q. For simplicity suppose that no path ever enters in the list-unnumhered lists. Then for each path $p\neq p_0$ the parent of p is defined to be the path containing the start of p in its inside.

Build the only simple path $\bar{p}$ in Q from p_0 to p_i and let $\bar{p}=(p_0', p_1', \dots, p_{i'}'=p_i)$. Then for j between 1 and i' the path p_j' belongs to some component with at least $n/3$ vertices among those components to which G is divided by M_j', where M_j' is the set of the vertices belonging to some of the paths p_0', $\dots$, p_{j-1}'. Furthermore, the algorithm of choosing p_j' among all regular paths with startvertex in p_{j-1}' (in Algorithm 3) is equivalent to that in Algorithm 4. Hence it is possible to apply Algorithm 2 (combined with Algorithm 3) to G so that at the j-th iteration of Algorithm 2 the path p_j' will be constructed. Therefore the set C' constructed by Algorithm 4 will have the structure of some of the graphs from Fig. 6 and the same upper bound of the partitioning set applies, namely $|C'|\leqq(7g+5)r+1$.

The proof for the case when list-unnumbered lists are used is similar, only the difinition of the tree Q is more lengthy.

If G is not biconnected, then consider the following two cases. If any of the bicomponents examined in Step 6.3 of SEARCH contains at least $n/3$ vertices, then C' will contain the corresponding articulation point and $|C'|=1$. If none of the bicomponents has $n/3$ vertices, then adding appropriate chosen artificial edges transforms G to a biconnected graph G' for which the above estimation can be obtained in the same manner.

So in all cases $|C'|\leqq(7g+5)r+1$.

Steps 10—15 of the main procedure are based on the proof of Theorem 2 in [8]. Using the technique of that proof, one can easily obtain the following estimation for the sets A, B, C constructed by Algorithm 4: $|A|\leq 2n/3$, $|B|\leq 2n/3$, $|C|\leq\sqrt{21g+15}\sqrt{n}$ (in Theorem 1 we have better estimation $|C|\leq\sqrt{12g+6}\sqrt{n}$).

Let us now examine the complexity of Algorithm 4.

If $m\geq 4n$ then the solution is constructed in $0(n)$ time in Step 1 of the main procedure.

Now suppose that $m<4n$. During the search in G carried out by the procedure SEARCH, each vertex is included in a path at most once, each edge is deleted at most once from any of the lists list-min, list-max (which lists have a total length $O(n+m)$). and each path is added and deleted at most twice from the lists list-unnumbered (with a total length $O(n)$). In any cycle of the procedure SEARCH for one step of the cycle either an edge is deleted from list-min or list-max, or a path is deleted from list-unnumbered, or an unexplored vertex is reached. All other operations that are not included in cycles require constant time per one call of SEARCH. Then the time for all calls of SEARCH is $O(n+m)$ plus time proportional to the number of calls of SEARCH. Since in different calls of SEARCH different paths are examined and the number of the paths does not exceed n, the number of the calls is $O(n)$. Then all the time required for all executions of SEARCH is $O(n+m)$. Obviously all other steps of Algorithm 4 require $O(n+m)$ time. Thus in the case $m<4n$ the complexity of Algorithm 4 is also $O(n)$.

We have just proved the following theorem:

Theorem 2. If G is any n-vertex graph of genus g, then Algorithm 4 finds in $O(n)$ time partitioning A, B, C of the vertices of G such that no edge joins a vertex in A with a vertex in B, $|A|$, $|B|\leq 2n/3$, $|C|\leq\sqrt{21g+15}\sqrt{n}$.

In some cases Algorithm 4 finds a separator G which is significantly smaller than it is guaranteed by Theorem 2. An interesting case is when G does not contain $K_{3,3}$ as a generalized subgraph (Fig. 4). In this case, according to the remark made in the analysis of Algorithm 2, all paths built during the search will divide the corresponding segments (i. e. the case illustrated in Fig. 2 (c) will not be possible). Then, no matter how great the genus of G is, the size of C in that case will not exceed the maximum size of a separator for planar n-vertex graphs from Theorem 2, i. e. $\sqrt{15n}$.

Theorem 3. *Let G be any n-vertex graph which does not contain $K_{3,3}$ as a generalized subgraph. Then there exists a partitioning A, B, C of the vertices of G, such that no edge joins a vertex in A with a vertex in B, $|A|$, $|B|\leq 2n/3$ and $|C|\leq\sqrt{15n}$.*

In [3] it is proved that for each large enough n there exists a graph with n vertices and at least $1/2\, n^{5/3}$ edges (thus with genus $\Omega(n^{5/2})$) which does not contain $K_{3,3}$ as a generalized subgraph.

Algorithm 4 was implemented in PLIOPT algorithmic language and tested on an ES-1022 computed. All the results proved its effectiveness.

For some applications of the separator theorems see [5], [7].

REFERENCES

1. A. V. Aho, J. E. Hopcroft, J. D. Ullman. The design and analysis of computer algorithms. Reading, Mass., 1974.
2. R. J. Lipton, R. E. Tarjan. A separator theorem for planar graphs. — *SIAM J. Appl. Math.*, **36**, 1979, 177-189.
3. H. N. Djidjev. A separator theorem. — *C. R. Acad. Buld. Sci.*, **34**, 1981, 643 — 645.
4. H. N. Djidjev. On the problem of partitioning planar graphs. — *SIAM J. Alg. Discr. Math.*, **3**, 1982, 229—240.
5. R. J. Lipton, R. E. Tarjan. Applications of a planar separator theorem. — *SIAM J. Comput.*, **9**, 1980, 615-627.
6. Х. Н. Джиджев. Теореми и алгоритми за разделяне на графи (Theorems and algorithms for partitioning graphs). Дис., С., 1983.
7. R. J. Lipton, D. J. Rose, R. E. Tarjan. Generalized nested dissection. — *SIAM J. Numer. Anal.*, **16**, 1979, 346-358.
8. H. N. Djidjev. A separator theorem for graphs of fixed genus. — *Serdica*, **11**, 1985, 319—329.
9. Ф. Харари. Теория графов. М., 1973.
10. J. E. Hopcroft, R. E. Tarjan. Algorithm 447: efficient algorithms for graph manipulations. — *Comm. ACM*, **16**, 1973, 372-378.
11. C. Jordan. Cours d'analyse. vol. I. Paris, 1893.
12. Г. Рингель. Теорема о раскраске карт. М., 1977.
13. W. G. Brown. On graphs that do not contain *a* Thomsen graph. — *Canad. Math. Bull.*, **9**, 1966, 281-285.

Centre for Mathematics and Mechanics
Sofia 1090 P. O. Box 373

Received 12.4.1984
Revised 30.7.1984