ISSN 1064-2269, Journal of Communications Technology and Electronics, 2017, Vol. 62, No. 6, pp. 653–662. © Pleiades Publishing, Inc., 2017.
Original Russian Text © K.Yu. Gorbunov, V.A. Lyubetsky, 2016, published in Informatsionnye Protsessy, 2016, Vol. 16, No. 2, pp. 223–236.

A Linear Algorithm for the Shortest Transformation of Graphs with Different Operation Costs

K. Yu. Gorbunov* and V. A. Lyubetsky**

Kharkevich Institute for Information Transmission Problems, Russian Academy of Sciences, Moscow, 127051 Russia
**e-mail: gorbunov@iitp.ru*
***e-mail: lyubetsk@iitp.ru*

Received June 16, 2016

Abstract—A novel time- and memory-efficient algorithm for solving the problem of finding the most economical (i.e., having the lowest overall cost) transformation of an arbitrary oriented graph representing a disjoint union of chains and cycles into another graph of the same type is proposed. The correctness of this algorithm (i.e., the fact that it always yields the minimum of the overall cost functional) and the linearity of the estimated memory and time of its operation are demonstrated.

Keywords: linear algorithm, oriented graph, chain, cycle, graph transformation, operation cost, combinatorial optimization

DOI: 10.1134/S1064226917060092

1. FORMULATION OF THE PROBLEM

The complete proof of correctness of a linear algorithm for solving the combinatorial optimization problem detailed below is given. A rigorous solution to this problem under general conditions discussed below was still not known.

An oriented graph with its connected components (without regard to orientation)—chains and cycles—is given. The graph edges are marked with positive integers (edge *names*) without repetitions. The problem of finding the most economical transformation of such an oriented graph into another one has been studied extensively at the heuristic level in the last two decades. The list of operations transforming such graphs into each other is fixed. More general types of graphs and an arbitrary list of operations may also be considered, but the indicated restrictions on graphs and the list of operations, which are given in Section 2, have taken shape for various (mostly practical) reasons over the long history of studies. A number (*cost*) is assigned to each operation. These numbers are positive and rational in applied problems, but in theory they may be considered natural. Every sequence of operations following each other, which starts from graph a and ends with a certain resulting graph b, has its *overall cost* (the sum of all operation costs in this sequence).

Formulation of the problem. Assume that we have two graphs a and b. The problem consists in finding the sequence of operations transforming a into b with the minimum value of the overall cost functional. Such a sequence is called the *shortest sequence* and its cost is the *shortest cost*.

It is assumed (although not proven) that the problem of finding the shortest sequence or the shortest cost for variable a and b and variable operation costs is NP-hard. It remains NP-hard if arbitrary (random) operation costs are fixed. Since only linear or, at least, low-degree polynomial algorithms are of practical importance, some restrictions need to be applied to cost. The simplest case is the case of equal costs. The problem with this restriction applied is called the *equal costs* problem.

Restrictions of a different type may also be applied to the considered problem: one and the same fixed set of names is required to be present in the sequence of operations transforming a into b (including a and b themselves). The problem with this restriction applied is called the *constant composition* (of edge names) problem. If this restriction is removed, the problem is called the *variable composition* problem.

A brief review of earlier studies can be found in [1, 2]. One possible numerical application of the proposed algorithm was discussed in [2]. An linear in time algorithm for constructing the shortest sequence for the equal costs problem was outlined in [1]. In this case, the sought-for sequence is called the *minimal* sequence.

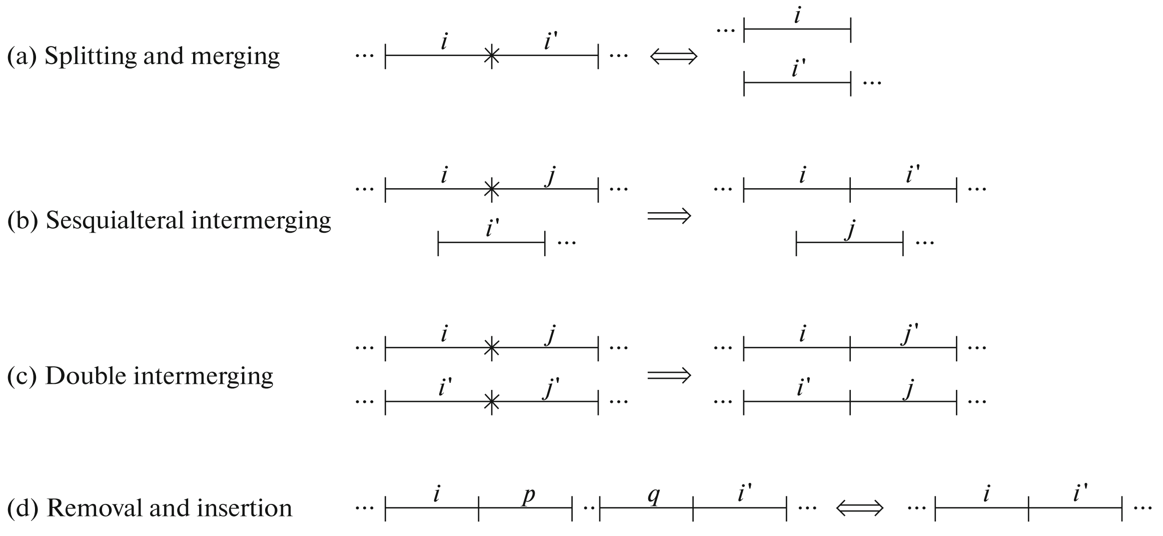
(a) Splitting and merging
(b) Sesquialteral intermerging
(c) Double intermerging
(d) Removal and insertion
i
i'
j
j'
p
q

Fig. 1. Operations that transform one graph of the indicated type into another graph of the same type.

2. LIST OF OPERATIONS AND AN AUXILIARY RESULT

The following six operations are allowed to be used in transformation of graph a into graph b. A node may be *split* (Fig. 1a) in order to break a chain into two chains or break a cycle. The inverse operation involves *merging* (matching) of two free ends of different chains or one and the same chain. *Sesquialteral intermerging* (Fig. 1b) consists in breaking a chain or a cycle and merging one of the newly formed ends with some other free end. *Double intermerging* (Fig. 1c) consists in breaking two chains (cycles) or a single chain (cycle) at two sites and merging of four newly formed ends in a different way. *Removal* of a section (several edges) of a chain or a cycle (Fig. 1d) may be performed if the edge numbers in this section are not found in graph b; the newly formed ends are merged. *Insertion* is the inverse operation of introduction of a section (with the edge numbers in this section lacking in graph a) into a chain or a cycle. The edge orientation is not used in operations. The first four operations are called *standard* operations and the remaining two are *additional* operations.

Let us recall the scheme of the algorithm from [1]. The notion of a *combined graph* $a + b$ of two oriented graphs a and b is introduced. Its nodes are designated as n_1 or n_2 for each edge with name n found in a and b; indices denote the head and the tail of an edge. These nodes in $a + b$ are called *ordinary* nodes. In addition, *blocks* (maximal (in inclusion) connected sections containing edges that belong only to a or only to b) serve as nodes in $a + b$. These nodes are called *singular* ones in $a + b$ and are designated with a set of names in the block. A combined graph $a + b$ has the following edges. An *ordinary* edge connects two ordinary nodes if their corresponding ends are merged (matched) in a or in b. A *singular* edge connects an ordinary node to a singular one if the end corresponding to an ordinary node in a or in b is merged with the end of the block corresponding to a singular node. Such an edge is designated as an a-edge or a b-edge, respectively. A *loop* in $a + b$ corresponds to a cycle that is also a block; in other words, the singular node of this block is connected to itself. An edge of a combined graph is called a *pendant* one if it has one singular end that is not connected to anything.

A combined graph contains chains and cycles, which are also called *components*. The *size* of a component is the sum of the number of ordinary edges and half the number of singular nonpendant edges in it. The size of a loop is 0, and the size of an isolated singular node (not a loop) is –1. It was proven in [1] that the initial problem is equivalent to the problem of transformation (reduction) of a combined graph to the *final* form (a graph comprised of cycles of length **2** without singular nodes and isolated ordinary nodes). The reduction of a combined graph is performed using similar operations with one exception: insertion is replaced by removal of a singular b-node. It is easy to verify that the proof of the equivalence of these two formulations of the problem in [1] remains valid if all standard operations have the same cost and the cost of insertion and removal is arbitrary.

3. REDUCTION OF A COMBINED GRAPH IN THE CASE OF CONSTANT COMPOSITION AND DIFFERENT OPERATION COSTS

Let us denote the cost of splitting, merging, sesquialteral intermerging, and double intermerging as c_1, c_1', $c_{1.5}$, and c_2, respectively. Two versions of the cost relationship are considered below: $c_2 \le c_1 \le c_1' \le c_{1.5}$ (*cyclic* version) and $c_1 \le c_1' \le c_{1.5} \le c_2$ (*linear* version).

We will seek the shortest sequence among all minimal sequences (i.e., solve the problem of constrained optimization). This sequence is called the *conditionally shortest* one. In the case under consideration, a combined graph contains cycles and chains with alternating *a*- and *b*-edges. If the composition is constant, we define quality $H(a + b)$ of a combined graph $a + b$ as the sum of the number of cycles and half the number of even chains in it. An even chain is a chain with an even number of edges or a chain with size 0; odd chains are neglected; in this section, the notions of size and length are considered interchangeable. Let graphs a and b have n edges each.

Lemma 1. *1. Each standard operation alters the quality of a combined graph by* 0 *or* ± 1.

2. An operation increasing the quality by 1 exists for a nonfinal graph.

3. *Graph $a + b$ is final if and only if $a = b$; $H(a + b) = n$ for a final graph.*

4. *Both unconditional and conditional problems for a and b are equivalent to the corresponding problems of reduction of combined graph $a + b$ to the final form.*

5. *There is a sequence of operations transforming $a + b$ to the final form with the quality increasing by exactly 1 at each step. Its length is $k = n - H(a + b)$.*

6. *The length of the minimal sequence of operations is k.*

7. *The sequences with each operation increasing the combined graph quality by 1 are the minimal ones for $a + b$. Their lengths are equal to k.*

Proof. 1. Let us consider standard operations one by one.

Double intermerging. If it is applied to a single cycle, one or two cycles are obtained, and the quality either remains unchanged or increases by 1. If it is applied to a single chain, one obtains either a single chain of the same length or a cycle with a chain having the same parity as the initial one. The quality either remains unchanged or increases by 1. If it is applied to two cycles, one cycle is obtained, and the quality decreases by 1. If it is applied to a cycle and a chain, one chain having the same parity as the initial one is obtained. The quality decreases by 1. If it is applied to two chains, two chains with the same overall length are obtained. The following transformations are possible here: two even chains are transformed into two even chains (the quality remains unchanged), two odd chains are transformed into two odd chains (the quality remains unchanged), two even chains are transformed into two odd chains (the quality decreases by 1), two odd chains are transformed into two even chains (the quality increases by 1), and even and odd chains are transformed into even and odd chains (the quality remains the same).

Sesquialteral intermerging. A free node is taken from a chain and is the outermost one. If it is applied to a chain and a node from it, a chain of the same length (the quality remains unchanged) or a cycle with a chain of the same parity (the quality increases by 1) are obtained. If it is applied to a chain and a node from another chain, two chains having the same overall length as the initial ones are obtained (all three scenarios of quality change found in the case of double intermerging are possible). If it is applied to a cycle and a node from a chain, a chain of the same parity is obtained. The quality decreases by 1.

Splitting. If it is applied to a cycle, a chain of an odd length is obtained (the quality decreases by 1). If it is applied to a chain, one obtains two chains that, when combined, are shorter than the initial chain by 1. The following transitions are possible: an even chain is transformed into even and odd chains (the quality remains unchanged), an odd chain is transformed into two odd chains (the quality remains unchanged), and an odd chain is transformed into two even chains (the quality increases by 1).

Merging. If it is applied to a single (necessarily odd) chain, a cycle is obtained (the quality increases by 1). If it is applied to two chains, a chain that is longer by 1 than the initial chains combined is obtained. The following transitions are possible: even and odd chains are transformed into an even chain (the quality remains unchanged), two odd chains are transformed into an odd chain (the quality remains unchanged), and two even chains are transformed into an odd chain (the quality decreases by 1).

5. The sequence is constructed through successive application of clause 2.

Simple proofs of other clauses are omitted.

The operation of our algorithms is illustrated in document no. 1 found at http://lab6.iitp.ru/-/graph_transformation_algorithms. These illustrations are not essential for formal understanding of the proofs and are referred to below without mention of the website link.

Let us characterize an exact linear algorithm for reduction to the final form in the cases of cyclic and linear cost relationships. The algorithm for the *cyclic version* involves three steps.

Step 1. If we have a cycle with its length strictly larger than 2, we break it up into two cycles by double intermerging. One of these new cycles has a length of 2 (see Fig. 2a).

Step 2. Each odd chain is transformed into a cycle by merging. Step 1 is then repeated (Fig. 2b).

Step 3. Each nonzero even chain is transformed into a cycle by means of sesquialteral intermerging. One chain end turns into a zero chain (Fig. 2c). Step 1 is then repeated.

The algorithm for solving the constrained problem in the linear version also involves three steps.

Step 1. It remains the same.

Step 2. The outermost node is separated from each odd chain by splitting, which yields an even chain (shorter by 1) and a zero chain (Fig. 3a).

Step 3. Each nonzero even chain is shortened by two edges by means of sesquialteral intermerging, and its outermost edges are closed into a cycle. This process continues until no nonzero chains are left in the combined graph (Fig. 3b).

If no one step is applicable to the examined combined graph, this graph is already in the final form. An empty sequence of operations is applied to it.

Theorem 1. *The indicated linear algorithms solve exactly the problem of constrained optimization for cyclic and linear cost relationships.*

Proof. The minimality of the obtained sequence follows from Lemma 1. It also follows from it that the algorithm is linear in time.

Let us prove that the obtained sequence is the shortest one. An a-chain in a combined graph is defined as an odd chain with its outermost edges labeled a. A b-chain is defined in a similar way. Let n_1 be the number of cycles in a combined graph and ℓ_1 be their overall length. The corresponding quantities for a-chains, b-chains, and even nonzero chains are n_a and ℓ_a, n_b and ℓ_b, and n_2 and ℓ_2, respectively. The overall cost in the obtained sequence for the cyclic version is

$$c_2\left[0.5(\ell_1 + \ell_2 + \ell_3) - n_1 - 0.5n_3 - n_2\right]$$
$$+ c_{1.5}n_2 + c_1 n_a + c_1' n_b,$$

and the cost of the linear version is

$$c_2(0.5\ell_1 - n_1) + 0.5c_{1.5}(\ell_2 + \ell_3 - n_3) + c_1 n_a + c_1' n_b.$$

This overall cost for combined graph G is *denoted* as $c(G)$. The shortest cost for reducing graph G to the final form is *denoted* as $C(G)$. We will show by induction in $C(G)$ that inequality $c(G) \le C(G)$ is satisfied for all graphs G. It then follows that $c(G) = C(G)$, which is the required result.

Since the number of nodes in graph G is fixed, the set of minimal costs is finite. Induction proceeds over the natural order in this cost set. If $C(G) = 0$, graph G is of the final form, and $c(G) = 0$.

Induction step. Let inequality $c(G) < C(G)$ be satisfied for all graphs G' with $C(G') < C(G)$. In order to prove it for G, we consider the reduction sequence for G. Let o be the first operation in it, $c(o)$ be its cost, and $o(G)$ be the result of application of o to G. It is sufficient to verify inequality

$$c(o) \ge c(G) - c(o(G)).$$

Since operation o increases the quality of G by 1, the number of cycles (or even chains) increases by 1 (or 2) after application of this operation. Let us analyze all possible cases.

1. Operation o is double intermerging that is applied to a cycle and breaks it up into two cycles (Fig. 2a). If this is the case, n_1 increases by 1, and other quantities in the formula for $c(G)$ remain unchanged. Equality $c(o) = c_2 = c(G) - c(o(G))$ is satisfied.

2. Operation o is double intermerging that is applied to a chain and cuts a cycle out of it (Fig. 4a).

A cycle and a nonzero chain having the same parity as the initial one are obtained. The value of n_1 increases by 1, and ℓ_1 increases as many as sum $\ell_2 + \ell_3$ decreases (for definiteness, by a certain p). Other quantities in the formula for $c(G)$ remain unchanged. Therefore, relations $c_2 = c(G) - c(o(G))$ and $c(G) - c(o(G)) = c_2 - 0.5c_2 p + 0.5c_{1.5}p \le c_2$ hold true for cyclic and linear versions, respectively.

3. Operation o is double intermerging that is applied to two odd chains (a-chain and b-chain; Fig. 4b). Two nonzero even chains of the same overall length are obtained. The values of n_a and n_b decrease by 1, n_3 decreases by 2, n_2 increases by 2, and ℓ_3 decreases as much as ℓ_2 increases. In the case of the cyclic relationship, $c(G) - c(o(G)) = -c_2 + 2c_2 - 2c_{1.5} + c_1 + c_1' = c_2 + c_1 + c_1' - 2c_{1.5} \le c_2$. If the cost relationship is linear, $c(G) - c(o(G)) = -c_{1.5} + c_1 + c_1' \le c_2$.

4. Operation o is sesquialteral intermerging that is applied to a chain and cuts a cycle off it (Fig. 2c). A cycle and (possibly zero) chain having the same parity as the initial one are obtained. The value of n_1 increases by 1, and ℓ_1 increases as many as sum $\ell_2 + \ell_3$ decreases (for definiteness, by a certain p). If the obtained chain is zero, n_2 also decreases by 1. In the cyclic case, $c(G) - c(o(G)) = c_2 \le c_{1.5}$ (nonzero case) and $c(G) - c(o(G)) = c_{1.5}$ (zero case). In view of the evident fact that $p \ge 2$, $c(G) - c(o(G)) = c_2 - 0.5c_2 p + 0.5c_{1.5}p \le c_{1.5}$ in the linear case.

5. Operation o is sesquialteral intermerging that is applied to two odd chains (a-chain and b-chain; Fig. 4c). Two even chains (one of them may be zero) of the same overall length are obtained. If both chains are nonzero, the reasoning is the same as in clause 3. If this is not the case (n_2 then increases by 1), $c(G) - c(o(G)) = -c_2 + c_2 - c_{1.5} + c_1 + c_1' = c_1 + c_1' - c_{1.5} \le c_{1.5}$ both for cyclic and linear versions.

6. Operation o is merging that is applied to an odd chain and transforms it into a cycle (Fig. 2b). Here, $c(o) = c_1$ if the chain is an a-chain, and $c(o) = c_1'$ if the chain is a b-chain. The value of n_1 increases by 1, n_3 and either n_a or n_b decrease by 1, ℓ_1 increases by $p \ge 2$, and ℓ_3 decreases by $p - 1$. In the case of the cyclic relationship, $c(G) - c(o(G)) = -0.5c_2 + c_2 - 0.5c_2 + c(o) = c(o)$. If the cost relationship is linear, $c(G) - c(o(G)) = -0.5c_2 p + c_2 + 0.5c_{1.5}(p - 2) + c(o) \le c(o)$.

7. Operation o is merging that is applied to an odd chain. Two even chains are obtained (Figs. 4a and 4d); either an edge labeled a in an a-chain or an edge labeled b in a b-chain is cut in the process. Here, $c(o) = c_1$ if the cut edge is labeled a, and $c(o) = c_1'$ if the edge is labeled b. When combined, the obtained even chains are shorter than the initial chain by 1. One of

these two chains or both of them may be zero. If both chains are nonzero, n_2 increases by 2, n_3 and either n_a or n_b decrease by 1, ℓ_2 increases by a certain p, and ℓ_3 decreases by $p + 1$. In the cyclic case, $c(G) - c(o(G)) = 0.5c_2 - 0.5c_2 + 2c_2 - 2c_{1.5} + c(o) \le c(o)$. If one of the obtained chains or both of them are zero, n_2 increases by 1 or remains unchanged. Correspondingly, 2 in the inequality is replaced by 1 or 0, and it remains valid. In the linear case, $c(G) - c(o(G)) = -0.5c_2p + 0.5c_2p + c(o) = c(o)$. □

4. REDUCTION OF A COMBINED GRAPH IN THE CASE OF VARIABLE COMPOSITION AND SPECIAL OPERATION COSTS

Combined graph $a + b$ and number ε, $0 \le \varepsilon \le 1$, are given. All operations (i.e., standard operations and removal of *a*- and *b*-nodes, which are singular nodes labeled *a* or *b*) are allowed. Let the cost of standard operations and *a*-removal be 1, and the cost of *b*-removal be $1 + \varepsilon$. The term *end* naturally applies to an edge end or an isolated node in a combined graph.

The proposed algorithm was tested on a computer in the general case, when the cost of *b*-removal is higher than the cost of other operations. The algorithm normally finds solutions close to the shortest sequence. This general case is not discussed in this study. However, in view of the possible heuristic application, certain explanations were added to the subsequent description of the algorithm. These explanations are not used in the proof given below.

Description of the Algorithm

Step 1. Remove singular *a*-loops.

Step 2. Cut out all ordinary edges not included into 2-cycles (i.e., cycles with size 2) and transform them into final 2-cycles by double intermerging (if the edge is not an outermost one, Fig. 5a), sesquialteral intermerging (if the edge is an outermost one, Fig. 5b), or merging (if the edge is isolated, Fig. 5c). If the cost of double intermerging is not higher than that of sesquialteral intermerging, double intermerging of all kinds is performed first. Otherwise, one should start with sesquialteral intermerging.

Step 3. Let us recall and generalize some of the definitions from [1]. An *a*-node is a singular node labeled *a* (the definition of a *b*-node is similar). An *odd* (*even*) chain is a chain of a odd (even) size. An *a*-chain is either an odd chain with its outermost nonpendant edges labeled *a* or an isolated *b*-node. The definition of a *b*-chain is similar. The chains and cycles left after steps 1 and 2 (with the exception of final 2-cycles and isolated ordinary nodes) are assigned to the following types: a cycle with an *a*-node and without a *b*-node is an "*a*-cycle" (the definition of a "*b*-cycle" is similar), and a cycle with both *a*- and *b*-nodes is assigned to type "cycle." A singular *b*-loop is assigned to type "loop." An *a*-chain is assigned to one of the following types: 1*a* (if it has one pendant edge), 2*a* (if it has two such edges), 2*a*' (if it is an isolated *b*-node), 3*a* (if it has no pendant edges, but contains both *a*- and *b*-nodes (its size is then larger than 1)), or 3*a*' (if it has neither pendant edges nor *b*-nodes (its size then equals 1)). The types assigned to *b*-chains are similar. Note that primed and unprimed types were introduced because of necessity to distinguish chains without *b*- or *a*-nodes (this was unnecessary in [1], and types 2*a*' and 3*a*' were included into 2*a* and 3*a*, respectively). An even chain is assigned to one of the following types: 1 (if it has a single pendant edge and both *b*- and *a*-nodes), 1' (if it consists of a single ordinary node and an incident *a*-node), 1'' (if it consists of a single ordinary node and an incident *b*-node), 2 (if it has nonpendant edges and two pendant edges), 2' (if it has just two pendant edges and no other edges), or 3 (if it has at least one edge and no pendant edges). Type 1 is divided into types 1_a (corresponding to chains with a pendant *a*-node) and 1_b (chains with a pendant *b*-node).

A single entry within steps 3 and 4 may include several successive transformations, which are separated by an equality sign. Within an entry, the types of chains (separated by the plus signs) prior to execution of an operation are indicated to the left of the equality sign, and the resulting chain type is indicated to the right of the equality sign. Isolated ordinary nodes and final 2-cycles are not indicated. For brevity, only the first equality is described; the descriptions of other equalities are similar. Types 2*a*, 3*b*, 1_b, and 2 are the combinations of types 2*a* and 2*a*', 3*b* and 3*b*', 1_b and 1'', and 2 and 2', respectively (this notation reduces the number of transformations). Type 1_c signifies a deferred choice between chains of type 1_a and 1_b (two possible results of the corresponding operation). Both results are preserved through to steps 4.15–4.24, where one result is selected in each pair.

Let us proceed with characterizing the algorithm. As a clarifying example, consider entry 3.2. Here, the indicated operation is applied successively to the pairs of chains of type 2*a* and 3*b*, and a 1_b chain is obtained as a result. Chains of type 2*b* and 3*a*, 2*b*' and 3*a*, 2*b* and 3*a*', and 2*b*' and 3*a*' are then transformed successively in a similar fashion into chains of type 1_a, 1_a, 1_a, and 1'. The other entries are organized in a similar way.

3.1. $1a + 1b = 1_c$. The outermost nonpendant edge (*exterior* edge) in one of the chains of type 1*a* or 1*b* is unmerged, and the corresponding singular node is merged with the outermost singular node of the other chain (sesquialteral intermerging). Two versions are shown in Fig. 6.

3.2. $2a + 3b = 1_b$, $2b + 3a = 1_a$, $2b' + 3a = 1_a$, $2b + 3a' = 1_a$, $2b' + 3a' = 1'$. The exterior edge in a 3*b*-chain is unmerged, and a singular node is merged with the outermost singular node of a 2*a*-chain (Fig. 7).

3.3. $2 + 3 = 1_c$. The exterior edge in a 3-chain is unmerged, and a singular node is merged with the outermost singular node of a 2-chain. A chain of type 1_a or 1_b (depending on which one of the two exterior edges is unmerged) is obtained (Figs. 8a and 8b).

3.4. $1b + 2a + 3 = 2 + 3 = 1_c$, $1a + 2b + 3 = 2 + 3 = 1_c$, $1a + 2b' + 3 = 2 + 3 = 1_c$. First $1b + 2a = 2$ (see description below), then $2 + 3 = 1_c$.

3.5. $1a + 3b + 2 = 3 + 2 = 1_c$, $1b + 3a + 2 = 3 + 2 = 1_c$, $1b + 3a' + 2 = 3 + 2 = 1_c$. First $1a + 3b = 3$ (see description below), then $2 + 3 = 1_c$.

3.6. $1a + 2 = 2a$, $1b + 2 = 2b$. The exterior edge in a $1a$-chain is unmerged, and a singular node is merged with the outermost singular node of a 2-chain (Fig. 9).

3.7. $1a + 3 = 3a$, $1b + 3 = 3b$. The outermost b-edge in a 3-chain is unmerged, and a singular node is merged with the outermost singular node of a $1a$-chain (Fig. 10).

3.8. $1a + 1a + 2b + 3b = 2 + 3 = 1_c$, $1a + 1a + 2b' + 3b = 2 + 3 = 1_c$, $1b + 1b + 2a + 3a = 2 + 3 = 1_c$, $1b + 1b + 2a + 3a' = 2 + 3 = 1_c$. First $1a + 2b = 2$ and $1a + 3b = 3$, then $2 + 3 = 1_c$.

3.9. $1a + 1a + 2b = 3a + 2b = 1_a$, $1a + 1a + 2b' = 3a + 2b' = 1_a$, $1b + 1b + 2a = 3b + 2a = 1_b$. First $1a + 1a = 3a$ (see description below), then $2b + 3a = 1_a$.

3.10. $1a + 1a + 3b = 1a + 3 = 3a$, $1b + 1b + 3a = 1b + 3 = 3b$, $1b + 1b + 3a' = 1b + 3 = 3b$. First $1a + 3b = 3$, then $1a + 3 = 3a$.

3.11. $1a + 1a = 3a$, $1b + 1b = 3b$. The outermost singular nodes of two $1a$-chains are merged (Fig. 11).

3.12. $1a + 2b = 2$, $1a + 2b' = 2$, $1b + 2a = 2$. The exterior edge in a $1a$-chain is unmerged, and a singular node is merged with the outermost singular node of a $2b$-chain (Fig. 12).

3.13. $1a + 3b = 3$, $1b + 3a = 3$, $1b + 3a' = 3$. The exterior edge in a $3b$-chain is unmerged, and a singular node is merged with the outermost singular node of a $1a$-chain (Fig. 13).

3.14. $2a + 2b + 3 + 3 = 2 + 3 = 1_c$, $2a + 2b' + 3 + 3 = 2 + 3 = 1_c$. First $2a + 2b + 3 = 2$, then $2 + 3 = 1_c$.

3.15. $3a + 3b + 2 + 2 = 3 + 2 = 1_c$, $3a' + 3b + 2 + 2 = 3 + 2 = 1_c$. First $3a + 3b + 2 = 3$, then $2 + 3 = 1_c$. The mentioned transitions are described below.

3.16. $2a + 3 + 3 = 1a + 3 = 3a$, $2b + 3 + 3 = 1b + 3 = 3b$, $2b' + 3 + 3 = 1b + 3 = 3b$. First $2a + 3 = 1a$, then $1a + 3 = 3a$.

3.17. $3b + 2 + 2 = 1b + 2 = 2b$, $3a + 2 + 2 = 1a + 2 = 2a$, $3a' + 2 + 2 = 1a + 2 = 2a$. First $3b + 2 = 1b$, then $1b + 2 = 2b$. See step 4 for description.

3.18. $2a + 2b + 3 = 2a + 1b = 2$, $2a + 2b' + 3 = 2a + 1b = 2$. First $2b + 3 = 1b$, then $1b + 2a = 2$.

3.19. $3a + 3b + 2 = 3a + 1b = 3$, $3a' + 3b + 2 = 3a' + 1b = 3$. First $3b + 2 = 1b$, then $1b + 3a = 3$.

Step 4. It is assumed that the cost of b-removal is higher than that of other operations (recall that step 4 is aimed at replacing costly b-removal with another operation). The algorithm depends on whether the cost of double intermerging is higher than that of sesquialteral intermerging. If this is the case, the operations in steps 4.1–4.24 are to be fulfilled successively. Otherwise, steps 4.1'–4.24' are applied. If sesquialteral intermerging and double intermerging have one and the same cost, one is free to choose steps 4.1–4.24 or 4.1'–4.24'. The operations at each step are first performed as many times as possible, and only then one proceeds to the next step. However, the proofs given below are applicable only to the case of equal cost of sesquialteral and double intermerging.

4.1. A "loop" + any type t with a b-node = type t. If t is not equal to $2a'$, a loop is "inserted" by double intermerging in a component of type t with subsequent matching of b-nodes. Otherwise, the same is done by sesquialteral intermerging (Fig. 14).

4.1'. The same as 4.1.

4.2. A "cycle" + any type t with a b-node and an a-node = type t. A cycle is inserted (by double intermerging with matching of two b-nodes) next to a b-node from a component of type t on the side of the a-node. The produced ordinary edge is cut out (Fig. 15).

4.2'. The same as 4.2.

4.3. $2a + 2b = 2 + 1'$. Sesquialteral intermerging with two nodes (the outermost a-node and the neighboring ordinary node) of a $2b$-chain cut off and the formed end merged to the outermost b-node of a $2a$-chain (Fig. 16).

4.3'. $2a' + 2b = 2 + 1'$.

4.4. $3a + 3b = 3$. The exterior edge in a $3a$-chain is unmerged, and a singular node is merged to the outermost ordinary node of a $3b$-chain (Fig. 17).

4.4'. $3a + 3b' = 3$.

Only the first case is considered below; the other cases are similar.

4.5. $2a + 3 = 1a$, $2b + 3 = 1b$. The exterior b-edge in a 3-chain is unmerged, and a singular node is merged with the outermost singular node of a $2a$-chain (Fig. 18).

4.5'. $2a' + 3 = 1a$.

4.6. $3a + 2 = 1a$, $3b + 2 = 1b$. The exterior edge in a $3a$-chain is unmerged, and a singular node is merged with the outermost singular node of a 2-chain (Fig. 19).

4.6'. $3a + 2' = 1a$, $3b' + 2 = 1b$.

4.7. $2a + 2a = 2a$, $2b + 2b = 2b$. The outermost singular nodes of two chains are merged (Fig. 20).

4.7'. $2a' + 2a = 2a$.

4.8. $3a + 3a = 3a$, $3b + 3b = 3b$. Two outermost ordinary nodes of chains are interconnected by an ordinary edge, which is then cut out (Fig. 21).

4.8'. $3b' + 3b = 3b$.

4.9. $1a + 2a = 1a$, $1b + 2b = 1b$. The outermost singular nodes of two chains are merged (Fig. 22).

4.9'. $1a + 2a' = 1a$.

4.10. $1a + 3a = 1a$, $1b + 3b = 1b$. Two outermost ordinary nodes of chains are interconnected by an ordinary edge, which is then cut out (Fig. 23).

4.10'. $1b + 3b' = 1b$.

4.11. $2a + 2 = 2$, $2b + 2 = 2$. The outermost singular nodes of two chains are merged (Fig. 24).

4.11'. $2a' + 2 = 2$, $2a + 2' = 2$, $2b + 2' = 2$.

4.12. $3a + 3 = 3$, $3b + 3 = 3$. Two outermost ordinary nodes of chains are interconnected by an ordinary edge, which is then cut out (Fig. 25).

4.12'. $3b' + 3 = 3$.

4.13. $2 + 2 = 2 + 1'$. Sesquialteral intermerging with two nodes (the outermost a-node and the neighboring ordinary node) of a 2-chain cut off and the formed end merged to the outermost b-node of another 2-chain (Fig. 26).

4.13'. $2' + 2 = 2 + 1'$.

4.14. $3 + 3 = 3$. The exterior a-edge in a 3-chain is unmerged, and the produced chain end is merged to the b-end of another 3-chain (Fig. 27).

4.14'. Empty operation.

4.15. $1_a + 1_a = 1_a$, $1_b + 1_b = 1_b$, $1_b + 1_c = 1_b$ ($c = b$ is set). The exterior edge in a 1_a-chain is unmerged, and a singular node is merged with the outermost singular node of another 1_a-chain (Fig. 28).

4.15'. $1'' + 1_b = 1_b$, $1'' + 1_c = 1_b$ ($c = b$ is set).

4.16. $1a + 1_b = 1a$, $1b + 1_a = 1b$, $1a + 1_c = 1a$ ($c = b$ is set). The exterior edge in a 1_b-chain is unmerged, and a singular node is merged with the outermost singular node of a $1a$-chain (Fig. 29).

4.16'. $1a + 1'' = 1a$.

4.17. $1a + 1_a = 1a$, $1b + 1_b = 1b$, $1b + 1_c = 1b$ ($c = b$ is set). The exterior edge in a $1a$-chain is unmerged, and a singular node is merged with the outermost singular node of a 1_a-chain (Fig. 30).

4.17'. $1b + 1'' = 1b$.

4.18. $2a + 1_b = 2a$, $2b + 1_a = 2b$, $2a + 1_c = 2a$ ($c = b$ is set). The exterior edge in a 1_b-chain is unmerged, and a singular node is merged with the outermost singular node of a $2a$-chain (Fig. 31).

4.18'. $2a' + 1_b = 2a$, $2a + 1'' = 2a$, $2a' + 1_c = 2a$ ($c = b$ is set).

4.19. $3a + 1_a = 3a$, $3b + 1_b = 3b$, $3b + 1_c = 3b$ ($c = b$ is set). The exterior edge in a $3a$-chain is unmerged, and a singular node is merged with the outermost singular node of a 1_a-chain (Fig. 32).

4.19'. $3b' + 1_b = 3b$, $3b + 1'' = 3b$, $3b' + 1_c = 3b$ ($c = b$ is set).

4.20. $2 + 1_a = 2$, $2 + 1_b = 2$, $2 + 1_c = 2$ ($c = b$ is set). The exterior edge in a 1_a-chain is unmerged, and a singular node is merged with the outermost singular node of a 2-chain (Fig. 33).

4.20'. $2' + 1_a = 2$, $2' + 1_b = 2$, $2 + 1'' = 2$, $2' + 1_c = 2$ ($c = b$ is set).

4.21. $3 + 1_a = 3$, $3 + 1_b = 3$, $3 + 1_c = 3$ ($c =$ b is set). The exterior edge in a 3-chain is unmerged, and a singular node is merged with the outermost singular node of a 1_a-chain (Fig. 34).

4.21'. $3 + 1'' = 3$.

4.22. $1_a + 1_c = 1_a$, $1b + 1_c = 1b$, $1a + 1_c = 1a$, $2b + 1_c = 2b$, $3a + 1_c = 3a$ ($c = a$ is set).

4.22'. Empty operation.

4.23. $c = b$ is set for the remaining chains of type 1_c, and $1_b + 1_b = 1_b$ is performed.

4.23'. Empty operation.

4.24. Chains with a nonpendant edge are closed into cycles by merging (chains type $2a$, $2b$, $3a$, and $3b$) or by sesquialteral intermerging with (chains of types 1_a, 1_b, 1_c, and 2) or without (chains of types $1a$, $1b$, and 3) singular nodes matching. When a type 1_c chain is closed, $c = b$ is set. When a type 2 chain is closed, the version with matching of two b-nodes is chosen (Fig. 35a), and an a-node is removed from the produced type 1' chain. Ordinary edges are cut out from cycles produced by closing chains of type $3a$ or $3b$. Step 4.2 is then repeated.

4.24'. The same as in 4.24.

Step 5. Isolated singular nodes and loops are removed. Singular nodes are removed from the remaining chains. 2-Cycles are cut out from cycles larger than 2 in size in such a way as to ensure matching of two b-nodes (as such, an a-node is included into a 2-cycle; Fig. 35b). Singular nodes are removed from 2-cycles.

The end of the description of the algorithm.

Let us prove the theorem of the minimality of the overall cost of the sequence of operations produced by the algorithm (i.e., the theorem of the exactness (correctness) of the algorithm).

Let B' be the number of b-cycles (cycles with a b-node and without a-nodes) in graph $a + b$. Let us recall the notation from [1]: B is the number of singular nodes in $a + b$; S is the sum of integer parts of half the number of edges (*length*) of maximal sections (*segments*) in $a + b$, which consist of ordinary edges, plus the number of odd (i.e., having an odd length) outermost segments minus the number of cyclic segments. An *outermost* segment is the segment located at the end of a chain (the case of a complete chain included). An *ordinary* pair consist of one of the standard operations, which does not alter the number of singular nodes, with its argument. In what follows, the term "ordinary" applies to an operation, while its argument is assumed to be given. The *defect* of a chain (or a cycle) is the minimum number of ordinary operations (the operation of cutting ordinary edges out at step 2 is not included) in a sequence that reduces it to the final form. Some operations with their arguments in the sequence may differ from ordinary ones; such operations are called *singular.* The dependence of the defect on the component type is given in [1]. Let us denote the sum of defects of the components of graph $a + b$ as D. The difference between the values of D calculated before and after step 3 is designated as P. Note that the number of singular operations in any sequence finalizing a combined graph equals the number of singular nodes in it. Thus, only the number of ordinary operations can be reduced. Since all operations at step 3 are singular ones, P equals the number of operations saved at step 3. Quantity ε was defined above. Let $C = B + S + D - P + \varepsilon(B' + 1)$.

Theorem 2. *The algorithm constructs a sequence of operations with its overall cost assuming one of the following three values*: $C - \varepsilon$, C, $C + \varepsilon$. *The minimum possible overall cost of a sequence of operations reducing graph* $a + b$ *to the final form also assumes one of these values. The algorithm running time is of a linear order of magnitude*.

Before proving the theorem itself, let us prove Lemmas 2–4.

Lemma 2. *With the exception of repeated execution of step 4.2 at step 4.24, it is impossible to perform any operation at step 4 (4.1–4.24 or 4.1'–4.24') again after it was performed for the first time; i.e., subsequent actions produce no components to which preceding operations are applicable*.

Proof. We go through the operations at step 4 and demonstrate for all of them that the result of each subsequent operation does not contain types present in the argument of the first operation, but absent in the argument of the second one. Since it is impossible to find a component of even one type of the arguments of an operation after its completion, repeated execution is excluded.

Lemma 3. After step 4, a total of 0, 1, or 2 connected components that have a b-node and are not the initial b-cycles are left.

Proof. By virtue of Lemma 2, cycles and chains without nonpendant edges, which allow none of the operations performed at step 4, remain after step 4. New b-cycles may emerge from type $3b'$ chains; in view of 4.8 (4.8') and Lemma 2, the maximum number of new b-cycles is 1. Thus, the components mentioned in Lemma 3 may be of the following types: $2a'$, $1''$, $2'$, cycle, and b-cycle. Applying the same reasoning, one can easily demonstrate that the maximum number of remaining components of the first three types is also 1. The only thing left to prove is that the maximum of two types out of the mentioned five may remain. Let us assume that this is not true. If the remaining three types do not contain a b-cycle, an operation may be performed between two of them (those that are not cycles). This contradicts Lemma 2. Otherwise, such an operation is possible between a type $3b'$ component, which gave rise to a b-cycle, and one of the three components that is not a cycle. Again, we arrive at a contradiction.□

Lemma 4. The number of ordinary operations in the algorithm is $S + D - P$.

Proof. Let us recall [1] that the minimum number of ordinary operations needed to reduce a component (after step 2) to the final form without the use of other components equals its defect. The algorithm in this study differs from the one in [1] in that it has an additional step (step 4). Each operation in step 4 is either a singular one, which does not alter the defect of the result relative to the overall defect of arguments, or an ordinary one, which reduces the defect by 1. Therefore, the number of ordinary operations in the algorithm remains unchanged (i.e., equals $S + D - P$).□

Proof of Theorem 2. A single operation of b-node removal is applied at step 5 to each component with b-nodes. By virtue of Lemma 3, the overall number of such operations is $B' + n$, where $n = 0$, 1, or 2. The number of singular operations is B. By virtue of Lemma 4, the overall cost of algorithm operations is $(1 + \varepsilon)(B' + n) + (B - B' - n) + (S + D - P) = B + S + D - P + \varepsilon(B' + n)$. The first statement of the theorem follows from this. The linearity of the algorithm time and the used memory is evident.

The second statement of the theorem is proven by induction for the minimal overall cost M of operations reducing a combined graph to the final form. A finite number of possible M values exist in any bounded interval, and we consider these values in the ascending order.

The basis of induction is evident. Let us characterize the induction step. It is sufficient to verify for any operation o applied to an arbitrary combined graph G that the cost of operation o $c(o)$ is not lower than $C(G) - C(o(G))$, where $C(G)$ is the C value defined in Theorem 2. If o does not alter the value of B', only $t = B + S + D - P$ may change in $C(o(G))$ relative to $C(G)$. The reasoning is then the same as in [1]; Theorem 5 validates the inequality. If o forces B' to increase, $C(G) - C(o(G)) \leq 1 - \varepsilon \leq c(o)$, since t may decrease by 1 at the most. Thus, it is sufficient to consider the case when o forces B' to decrease. This analysis involves the following five steps.

1. Operation o is singular node removal. If o forces B' to decrease (evidently, by 1), o removes a b-node, and $C(G) - C(o(G)) \leq 1 + \varepsilon = c(o)$.

2. Operation o is merging. It may not reduce the value of B'.

3. Operation o is splitting. One should consider the case when B' decreases by 1 and a b-cycle is split. Since an odd number of ordinary edges is present in a b-cycle between b-nodes, either a type $3b$ chain or a type $2a$ chain (according to the classification from [1] (Lemma 6) for components of the general form) is produced by splitting. In the former case, S remains unchanged, and D increases by 1; in the latter case, the

Results of merging the ends of two chains

0*a*	0	1*a*	1*a*'	2*a*	3*a*	1	1'	2	3
(0)	(0, +1)	(0, –1)	(0, +1, –1)	(0, –1)	(0, +1, –1)	(0, –1)	(0, +1, –1)	(0, –1)	(0, +1, –1)
[0]	[0, –1]	[–1, 0]	[–1, –2, 0]	[–1, 0]	[0, –1, +1]	[–1, 0]	[0, –1, +1]	[–1, 0]	[–1, –2, 0]
2*a*	1	1*a*	2*a*	2*a*	1*a*	1	2	2	1

effect is reversed. In both cases, B remains unchanged, and P either also remains unchanged or increases by 1. Thus, $C(G) - C(o(G)) \le \varepsilon < c(o)$.

4. Operation o is sesquialteral intermerging. One should consider the case when B' decreases by 1, a b-cycle is split, and the obtained chain is merged with another chain. It was demonstrated at the previous step that t does not decrease when a b-cycle is split. Therefore, it is sufficient to demonstrate that, if t remains unchanged (i.e., P increases by 1) when a b-cycle is split, it also does not decrease when two chains are merged.

Let us assume that a type 2*a* chain is produced as a result of b-cycle splitting. The possible types of chain merging are listed in the table. The first chain is a type 2*a* one, and the end type of the second chain is indicated in the column header (the notation is the same as in [1]).

Searching through the table cells, we determine the variation of P corresponding to reduction t. For example, this reduction corresponds to preservation of P in column 1*a* (the second chain has type 1*a*). This is impossible, since a type 1*a* chain is again produced by merging, and, if P increases on emergence of a type 2*a* chain, P should decrease on its disappearance. Likewise, column 1*a*' tells us that a type 2*a* chain is produced, which is equivalent to elimination of a type 1*a* chain by merging chain types 2*a* and 1*a*. However, if P decreases upon disappearance of a type 2*a* chain, it should certainly decrease upon disappearance of a type 1*a* chain. Columns 2*a*, 1, 2, and 3 are analyzed in a similar fashion. Merging operation $2a + 3a = 1a$ corresponds to column 3*a*. This operation does not result in an increase in P: if we assume that P does increase, it will increase by 2 following the substitution of a type 3*a* chain with a type 1*a* chain (the result of application of operation o), and this is impossible. Merging operation $2a + 1 = 2$ corresponds to column 1'. This operation does not result in an increase in P: if we assume that P does increase, it will increase by 2 upon emergence of a type 2 chain (the result of application of operation o), and this is impossible. The case when a type 3*b* chain is produced by splitting a b-cycle is considered in a similar fashion.

5. Operation o is double intermerging. Let us consider possible cases when B' decreases.

5.1. Operation o is applied to two b-cycles and produces a single b-cycle. Since an odd number of ordinary edges is present between b-nodes in b-cycles (or in one b-cycle, which is sufficient), the value of $B + S$ does not decrease. D and P remain unchanged, and B' decreases by 1. The required result is evident.

5.2. Operation o is applied to two cycles. One of them is a b-cycle, and the other contains both types of singular nodes. The reasoning is the same as in the above entry.

5.3. Operation o is applied to a b-cycle and a chain; a cycle is inserted into a chain. It needs to be demonstrated that $t = B + S + D - P$ does not decrease in this case.

5.3.1. If singular nodes are present in a chain on both sides of the cut, the chain type remains unchanged. Therefore, D and P also remain unchanged. Since the segment (with ordinary edges) adjacent to the cut in a b-cycle is odd, $B + S$ does not decrease, which is the required result.

5.3.2. Let us assume that singular nodes are present in a chain only on one side of the cut. If the parity of the outermost chain segment remains the same, the chain type does not change, and the above reasoning applies. If an even outermost segment is substituted with an odd one, B remains unchanged (we take into account the fact that the segment in a b-cycle is odd), and S increases by 1. The following chain type changes are then possible: $3a \to 1a$, $1 \to 2$, $1a \to 2a$, and $3 \to 1$. In the first two cases, D remains the same, and P either also remains unchanged or increases by 1. In the last two cases, D decreases by 1, and P either remains unchanged or decreases by 1. Thus, t does not decrease. If an odd outermost segment is substituted with an even one, B and S naturally remain unchanged. The following chain type changes are then possible: $1a \to 3a$, $2 \to 1$, $2a \to 1a$, and $1 \to 3$. In the first two cases, D remains the same, and P either also remains unchanged or decreases by 1. In the last two cases, D increases by 1, and P either remains unchanged or increases by 1. Thus, t does not decrease.

5.3.3. A chain has no singular nodes. It is more convenient to consider operation o' (the inverse of o), which cuts a cycle out of a chain, in the case when all singular nodes are located between the cuts. It is sufficient to prove that operation o' cannot alter t by more than 1, and if two singular nodes are not matched and all segments in the obtained cycle are odd, t does not increase. Let us analyze two cases.

1. Two singular nodes are matched; i.e., B decreases by 1. Two outermost chain segments are

then matched, and an additional ordinary edge emerges. If both segments are even, S increases by 1. The chain type is then $3a$ or $3b$; when it disappears, D decreases by 1 and P either remains unchanged or decreases by 1. If one segment (or both segments) is odd, S remains unchanged. The chain type is then $2a$, $2b$, or 1; when it disappears, D remains the same and P either remains unchanged or decreases by 1. Thus, t changes by no more than 1 in each case.

2. Two singular nodes are not matched; i.e., B remains unchanged. Two outermost chain segments are then transformed into two segments of the same overall length (one (outermost) segment in a chain and another segment in a cycle). Let us analyze the possible scenarios.

2.1. A pair of even segments is transformed into a pair of even or odd segments. Quantity S remains unchanged. The chain type was $3a$, $3b$, or 3; when it disappears, D decreases by 1, and P either remains unchanged or decreases by 1. Thus, t either remains unchanged or decreases by 1.

2.2. A pair of odd segments is transformed into a pair of even or odd segments. Quantivy S decreases by 1. The chain type was $2a$, $2b$, or 2; when it disappears, D remains the same, and P either remains unchanged or decreases by 1. Thus, t either remains unchanged or decreases by 1.

2.3. A pair of segments of different parity is transformed into a pair of segments of different parity, where the even segment is in a cycle. Quantity S remains unchanged. Since an even segment emerges in a cycle, outermost singular nodes of a chain had different labels. The chain type was then $1a$ or $1b$; when it disappears, D decreases by 1, and P either remains unchanged or decreases by 1 or 2. Thus, t changes by no more than 1.

2.4. A pair of segments of different parity is transformed into a pair of segments of different parity, where an odd segment is in a cycle. Quantity S decreases by 1. Since an odd segment emerges in a cycle, outermost singular nodes of a chain had the same labels. The chain type was then 1; when it disappears, D and P remain unchanged. Thus, t decreases by 1.

ACKNOWLEDGMENTS

This study was supported by the Russian Science Foundation, project no. 14-50-00150.

REFERENCES

1. K. Yu. Gorbunov and V. A. Lyubetsky, "Linear algorithm for minimal rearrangement of structures," Probl. Inf. Transmis. **51**(1), 55–72 (2017).
2. V. A. Lyubetsky, R. A. Gershgorin, A. V. Seliverstov, and K. Yu. Gorbunov, "Algorithms for reconstruction of chromosomal structures," BMC Bioinformatics **17**, 40 (2016).

Translated by D. Safin