ISSN 0032-9460, Problems of Information Transmission, 2017, Vol. 53, No. 1, pp. 55–72. © Pleiades Publishing, Inc., 2017.
Original Russian Text © K.Yu. Gorbunov, V.A. Lyubetsky, 2017, published in Problemy Peredachi Informatsii, 2017, Vol. 53, No. 1, pp. 60–78.

Linear Algorithm for Minimal Rearrangement of Structures[1]

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

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

Received December 29, 2014; in final form, April 25, 2016

Abstract—We propose a linear time and linear space algorithm which constructs a minimal sequence of operations rearranging one structure (directed graph of cycles and paths) into another. Structures in such a sequence may have a varying number of edges; a list of operations is fixed and includes deletion and insertion of a fragment of a structure. We give a complete proof that the algorithm is correct, i.e., finds the corresponding minimum.

DOI: 10.1134/S0032946017010057

1. INTRODUCTION

In Section 2 we present formulations and definitions related to the problem considered in the paper and problems close to it. There are many publications devoted to such problems, among which we note the fundamental work [1]. Below we only mention some papers most close to the present paper; see also [2]. These problems are evoked by biological and medical subjects, so terms from these areas are commonly used in their settings. However, we consider the mathematical component of these problems only; the obtained solutions can also be applied, for example, in engineering issues.

In [3] there were proposed operations for rearranging chromosome structure (see Section 2); in what follows they are referred to as *standard* operations and are a part of our more general set of operations. In [3] there is also given an algorithm to compute the number of operations in the *minimal* (in the number of operations) sequence transforming one structure into another if the structures consist of paths ("linear chromosomes") only; the algorithm is almost linear in time, and no estimate for the runtime is presented. These operations being applied to paths only correspond to reversals, translocations, fusions, and fissions previously considered in [4]. In the case where all intermediate structures also consist of linear chromosomes, an algorithm for computing the minimum number of operation is also given in [4]. The general case of genomes having equal gene content and *equal cost of operations* was solved in [5] using ideas from [3, 4].

In the case of *genomes with unequal gene content* one needs *additional* operations: deletion and insertion of strings of unique genes, which were proposed in [6, 7].

Papers [6–8] uses the notion of an adjacency graph: its vertices are adjacencies of extremities (heads and tails) of genes that belong to both structures and ends of the original paths. Two adjacencies from different structures are joined by an edge if they represent the head or tail of the same gene. Furthermore, an end of the original path is assumed to be adjacent to an "empty extremity" (telomere); between adjacent extremities of *common* genes of two structures there can

[1] The research was carried out at the Institute for Information Transmission Problems of the Russian Academy of Sciences at the expense of the Russian Science Foundation, project no. 14-50-00150.

be a string of genes belonging to only one structure (*unique* genes), and in this case it is a label of this vertex. Such a graph is graph obviously different from the graph defined below. In these papers there was proposed a linear time algorithm constructing a minimal sequence of rearrangements of one structure into another using the same operations as in the present paper. In [6, 7], costs of all operations equal 1; in [8] the cost of a standard operation is 1, and the costs of insertion and deletion are the same and less than 1. It is unclear whether the algorithm of those papers can be related to our algorithm. The authors are unaware of where a proof of correctness of the algorithm from [6–8] is given; remarks contained in these works do not allow to obtain such a proof.

In [9–11] a linear algorithm is proposed based on completing both original structures with unique genes. Thus, the problem reduces to the case of equal gene content, and the total number of genes increases by $k+t$, where k and t are the numbers of unique genes in the original structures. A graph is used which additionally comprises two extremities of each unique gene, which results in enlarging the graph that their algorithm works with as compared with the graph proposed in [3, 12]. The graph and algorithm proposed in [2, 12] are distinct from those in [9–11]. In [11] there is described a generalization of the algorithm of [9, 10] to the case where all chromosomes are cyclic, the cost of standard operations is 1, and the costs of insertion and deletion are equal. The proof of correctness given in [11] contains, in our opinion, some gaps.

Note the works [13, 14]. In [13] there is considered a generalization of the double-cut-and-join operation to the case of an arbitrary number of breaks with gluing the obtained fragments; other operations are not considered. Algorithms are given for finding the minimal sequence in the case where structures have equal gene content and only cyclic chromosomes. One of the algorithms has runtime $O(n^{0.5k-2}) + O(n)$, where n is the size of original structures and k is the maximum number of breaks; another one has runtime linear in n but requires preprocessing time exponential in k. A graph is used which in this special case coincides with our graph. In [14], the problem is considered of finding the length of a minimal sequence of rearrangements taking on structure to another provided that they have equal gene content and only linear chromosomes. In this paper, no algorithm is proposed but explicit lower bounds on the length are given, as well as results showing that they are rather tight. Here the problem is solved by reduction to the case of structures with only cyclic chromosomes; for that, gluing of gene endpoints is used. The authors were unaware of the works [13, 14] and, unfortunately, did not consider such rearrangements.

In the present paper, for a directed graph consisting of an arbitrary set of paths and cycles ("chromosomes") and all the above-mentioned operations except for the generalized rearrangements from [13, 14], we for the first time (as far as we know) give a complete proof of correctness of a linear time and linear space algorithm that we propose in the case of unequal gene content and equal operation costs. The case of different operation costs is considered by the authors in [2].

2. PROBLEM AND BASIC NOTIONS

2.1. Problem Statement

Let us give all necessary definitions. A *structure* is a finite set of directed graphs ("components"), each component being either a linear path or a cycle, including a loop. An edge is referred to as a *gene*; a component, as a *chromosome*. A gene is assigned with its *number*, a natural number, which, in general, may occur repeatedly. In the present paper, such repetitions are forbidden; the case with possible repetitions ("paralogs" of genes) is considered in [2, 15].

Usually, there are many components. All the structures can be considered as a directed graph of a special form (sometimes called a CC-graph).

Let two structures, a and b, be given, and let a set of operations that transform one structure into another be fixed. Every operation is assigned with a positive rational number, its *cost* (sometimes called the *weight*). In the present paper all operation costs are assumed to be equal to 1 (or, equivalently, to be the same).

The *problem* consists in finding the *shortest* sequence of operations transforming a into b. Clearly, "the shortest" means a sequence with the minimal total cost of all operations. This minimal cost will be referred to as the *shortest cost*. In the case considered here, we speak about the *minimal* sequence and call the shortest cost the *minimum length*. *Reduction* of a to b means using the shortest sequence of operations transforming a into b; this sequence is said to be *reducing*. If $a = b$, then the reduction is formally made by an "empty sequence of operations," which is assigned with zero cost and length.

Usually, the set of operations contains an inverse to each of the operations; therefore, it makes no difference whether we reduce a to b or b to a. Costs of a direct and inverse operations may be different.

This problem in a particular case where a and b have the same gene content and costs of all operations equal 1 is solved in [5] using a more complicated algorithm than in [12].

Below, on a strictly mathematical level, we solve this problem under unequal gene content. Some technical details of computations used in proofs of Theorems 1, 2, and 5 are deferred to the Appendix given at `http://lab6.iitp.ru/ru/pr_chromo/`. The obtained result was presented at conferences [15–18], including the description of the algorithm and proofs, and is also described in [2].

Let us mention results that take place in the case of unequal costs. In this case, specific conditions on the costs must be imposed, since the problem is assumed to be NP-complete and in the general form cannot be solved by not only linear but also any polynomial algorithm. At the above-mentioned conferences there were presented linear time and linear space algorithms for solving this problem in the case of unequal costs; see also [2]. The first two variants of the costs are as follows: $c_2 \le c_1 \le c_1' \le c_{1.5}$ (*cyclic*) and $c_1 \le c_1' \le c_{1.5} \le c_2$ (*linear*); here we indicate the relations between the costs c_1 of a fission, c_1' of a fusion, $c_{1.5}$ of a 1.5-break, and c_2 of a 2-break; the gene content is assumed to be equal. The third variant is as follows: the cost of an insertion is $1 + \varepsilon$ with $0 \le \varepsilon \le 1$, and the cost of the other operations is 1; gene content is unequal. For the third variant, the corresponding algorithm constructs a sequence of total cost differing from the minimum possible by at most ε.

2.2. Notions of a Common Graph

Our original approach is based on the proposed notion of a common graph; in fact, we also use the definition of its quality [12].

We denote gene extremities by i_j, where i_1 is the tail of gene i and i_2 is its head; a loop in the structure means a gene whose head coincides (is adjacent) with the tail. We first define the notion of a common graph in the particular case where structures a and b have *equal gene content*, i.e., their edges are assigned with the same collection of natural numbers without repetitions, for instance, all numbers from 1 to n. A *common graph* of structures a and b is a (now undirected) graph $a + b$ whose vertices are extremities i_j of all genes from a (or from b, since the sets of genes and extremities are the same), and edges join the extremities adjacent in a or in b; each edge is labeled by the name of the structure where adjacency occurs, i.e., by names a or b. Some edges in $a + b$ can be parallel: one edge from a, and the other from b; these are cycles of length 2, which will be referred to as 2-*cycles*.

It is easily seen that components of the common graph $a + b$ are alternating (a, b)-paths and (a, b)-cycles. As will be seen below, the algorithm for finding the minimum length does not use labels i_j assigned to the vertices.

Now consider a nontrivial definition of $a + b$ for the case where a and b have *unequal gene content*. We refer to genes belonging to both structures a and b as *common*, and the others, as *unique*. Inclusion-maximal fragments in a or b consisting of unique genes only are called *blocks*; the *name* of a block is the set of names of genes contained in it. Vertices of the graph $a + b$ are extremities i_j of all common genes (*ordinary* vertices) and *special* vertices, which correspond to blocks and are labeled with their names. We identify a special vertex with its name. Furthermore, we *label* a special vertex by name a or b depending on the belonging of the block.

Define edges in $a + b$. Ordinary vertices are joined if they are adjacent in a or b (as in the case of equal gene content). If a block with name A is a cyclic component (including a loop), then we draw a loop at the isolated vertex A. This loop is said to be *special*. If a block A is a linear component, then A remains an isolated vertex. Isolated special vertices are assumed to be odd paths of *length* -1, and isolated ordinary vertices are even paths of *length* 0.

If a linear block A is between two extremities of common genes (i.e., the block is neither a whole path nor a whole cycle), then these extremities are joined by edges with this special vertex A. If a linear block A is joined in the structure with only one extremity of a common gene, then this extremity in the common graph is joined by an edge with the special vertex A (a *hanging* edge). An edge incident to two ordinary vertices is said to be *ordinary*; an edge incident to a special vertex is said to be *special*. $\triangle$

By a *final* (or *final-form*) graph we call a common graph with ordinary edges consisting of 2-cycles and isolated ordinary vertices.

An operation is performed over one or two component(s) (chromosomes) belonging to the structure. An operation over a pair of structures $\langle a, b\rangle$ is defined as an operation over one of the components; the other component remains unchanged. An operation over a pair $\langle a, b\rangle$ is naturally transferred to their common graph, the name of the operation is kept the same; it is important that the corresponding diagram is commutative.

The *considered problem in the language of a common graph* is formulated as follows. Find the shortest sequence which reduces a given common graph $a + b$ to a final form; for the cost of an operation applied to b we take the cost of the inverse operation. Operations allow to distinguish application to a or to b.

It is convenient to store a common graph $a+b$ as an array M with indices varying from $-n$ to n; negative indices correspond to tails of the similar genes, and positive to heads, in the structures a and b. The value $M[i]$ is a pair of indices of extremities to which the extremity i is adjacent in a and b (and the value 0 if it is not adjacent). Such storing ensures linear time and space of the algorithm for constructing the graph $a + b$ given the structures a and b, as well as a fast screening of its components and backward transition from an operation over $a+b$ to an operation over a or b.

2.3. Original Operations over a Structure and a Common Graph

We consider the following *operations* over a structure: cutting two adjacencies and rejoining the four extremities in another way (double-cut-and-join, or a 2-break); cutting two adjacencies and joining one extremity with some *free* (i.e., nonadjacent) extremity (1.5-break); cutting an adjacency, thus forming two free extremities and the inverse operation of merging two free extremities (fission and fusion, respectively). These operations, sometimes referred to as *standard*, were proposed in [3] and more accurately in [5]. In the case of unequal gene content, one must also use *additional* operations: deletion or insertion of a fragment of unique genes. In a form convenient for us, they are described in the beginning of Section 3.1.

Accordingly, to the graph $a + b$ the following *operations* are applied similar to standard ones. 2-break: deletion of two identically labeled edges and joining their four endpoints by two new nonincident edges with the same labels; 1.5-break: deleting an edge and joining (by an edge with the same label, say a) one of its endpoints with an ordinary vertex not incident to an edge with this label, or with a special vertex of degree at most 1 with this label; fusion: adding an edge (say with label a) between vertices, and ordinary one not incident to an edge with label a, or a special of degree at most 1 with label a; fission: deleting an edge. If some operation results in occurrence of an edge with special endpoints (both belonging to a or both to b), it is replaced with a special vertex, whose name is the union of names of the endpoints (the number of special vertices reduces by 1).

3. REDUCTION OF STRUCTURES IN THE CASE OF UNEQUAL GENE CONTENT AND EQUAL OPERATION COSTS

Recall that gene contents in two structures a and b can be different, and besides the four standard operations there can be used additional operations of insertion and deletion. *Cost of all operations are the same.* It is required to find a minimal ("reducing") sequence from a to b. Recall (in a somewhat more general form): in an arbitrary structure, an inclusion-maximal fragment consisting of special genes all of them belonging to a or all belonging to b is called a *block*.

It is convenient to think that the problem consists in finding a structure c to which a and b are independently reduced by sequences minimal in aggregate (then a can be reduced to b through c), which is equivalent to reducing $a + b$ to $c + c$, and the latter graph is already of a final form.

Exterior extremities/genes are adjacent to the block. Recall that an ordinary edge joins ordinary vertices, a special edge joins either a special extreme vertex in a path with an ordinary vertex ("hanging edge") or an interior special vertex in a path or a cycle with an ordinary one.

The *length* of a path or a cycle is the number of ordinary edges in it plus half the number of special nonhanging edges. Odd (even) paths are paths of odd (even) lengths.

A common graph is said to be *special* (of a *special form*) if it contains no loops and if all edges that do not enter 2-cycles are special. 2-cycles may contain special and ordinary edges, and isolated vertices can be both special and ordinary.

Reducing a graph to a special form is the first step of our algorithm, and the second consists in reducing the special graph to a final form. Essential is that it is better to delete special edges jointly, thus saving on the number of operations, which is desirable; and ordinary edges should be deleted one by one in any order.

Operations are performed successively, starting from the given structure; therefore, each operation is considered together with the structure to which it is applied. We say that an operation is *ordinary* (*special*) if performing it does not change (strictly reduces) the number of special vertices. From the definitions in Section 2.3 it is seen that a special operation reduces this number precisely by 1. Indeed, performing a 2-break, 1.5-break, or fusion results in no more than one new edge joining special vertices, which merge thereupon into one. The deletion operation also deletes only one special vertex.

The idea of the algorithm is as follows: first we apply ordinary operations only and delete ordinary edges until the fraction of special edges becomes the maximal; a graph of a special form appears; we mainly apply to it special operations to get rid of special vertices; a graph of a final form appears.

3.1. Additional Deletion and Insertion Operations. Possibility of Doing without One of Them

A *subblock* is a connected fragment of a block (i.e., the maximality condition is dropped). We call it an *a-subblock* if it consists of genes that belong to a only. A b-subblock is defined similarly.

The *deletion operation* is deleting an a-subblock; it is admissible if the subblock:

(a) was strictly inside a linear or cyclic chromosome, then the two endpoints of the exterior genes become joined;
(b) was at the end of a linear chromosome, then the endpoint of the exterior gene becomes free;
(c) was a separate chromosome.

The *insertion operation* is inserting a b-subbslock, which is admissible if the subblock is joined:
(a') strictly inside a linear or cyclic chromosome, the insertion position being cut;
(b') at the end of a linear chromosome if the subblock is a linear path;
(c') as a separate chromosome.

Deletions and insertions defined in items (a) are said to be *rigid*, those defined in (b) are called *semirigid*, and those in (c) are *free*.

An *a-block* consists of genes of the structure a. An insertion operation *splits* an a-block if it is linear and the subblock is inserted between two of its genes. 2-break, 1.5-break, and fission operations *split* an a-block if they comprise one or two cut(s) inside a linear a-block or two cuts in a cyclic a-block. A deletion operation *splits* an a-block if only a part of it is deleted.

Theorem 1. *Among minimal sequences reducing a to b there exists a sequence in which*
1. *No operation splits an a-block*;
2. *All deletions are made before all insertions.*

Proof. Some actually obvious computations are deferred to the Appendix at `http://lab6.iitp.ru/ru/pr_chromo/`.

For brevity, we will refer to a-blocks as *blocks*, since no other blocks will be considered. Denote by $r(a, b)$ the minimum length for structures a and b.

Lemma 1. *If a structure d' is obtained from a structure d by deleting one gene g from a block, then $r(d, b) \geq r(d', b)$.*

Proof. Use induction on $r(d, b)$. If $r(d, b)$ equals 1, then b is obtained from d by one deletion, and $r(d', b) = r(d, b)$. Otherwise, consider the fist operation in the minimal sequence from d to b; denote it by o. It corresponds to an operation o' over d' (maybe empty) such that the structure $o'(d')$ either coincides with the structure $o(d)$ or is obtained from it by deleting the gene g. By the induction hypothesis, $r(o(d), b) \geq r(o'(d'), b)$. Hence, $r(d, b) = r(o(d), b) + 1 \geq r(o'(d'), b) + 1 \geq r(d', b)$. △

Lemma 2. *If a structure d' is obtained from a structure d by adding to a block a gene g not belonging to b, then $r(d, b) = r(d', b)$.*

Proof. Use induction on $r(d, b)$. If $r(d, b) = 1$, then b is obtained from d and d' by a deletion, and the claim is proved. Otherwise, consider the first operation in the minimal sequence from d to b; denote it by o. It corresponds to an operation o' over d' such that $o'(d')$ is obtained from $o(d)$ by adding g to some block. By the induction hypothesis, $r(o(d), b) = r(o'(d'), b)$. Hence, $r(d, b) = r(o(d), b) + 1 = r(o'(d'), b) + 1 = r(d', b)$. △

An operation which does not split a block will be referred to as *integral*. A sequence of integral operations will be called *integral*. A *contraction* of a block is deleting genes from it, and an *extension* of a block is adding genes to it that do not belong to b. A structure d' is a *simplification* of a structure d if d' is obtained from d by contraction or extension of blocks (i.e., a block cannot appear in d' in a place where in d there were no blocks).

Corollary 1. *There exists a minimal sequence of operations from a to b all deletions in which are integral.*

Proof. In an arbitrary minimal sequence, consider the first nonintegral deletion. Denote by d the structure obtained after applying this operation. Replace this operation by an integral deletion: delete the whole block and obtain a structure d'. By Lemma 1 we have $r(d, b) \geq r(d', b)$. Thus, we can lengthen the head of the minimal sequence that does not contain nonintegral deletions. △

Corollary 2. *There exists a minimal sequence of operations from a to b all deletions and insertions in which are integral.*

Proof. By Corollary 1 there exists a minimal sequence S all deletions in which are integral. Consider the first nonintegral insertion in it which splits some block r. Denote by d the structure obtained after applying this insertion. Now change the place of insertion: insert the same fragment at any end of the block r. The obtained structure d' is a simplification of d, since it is obtained from it by deleting one block (any part of the split block) and extending another block (the other part of the split block). By Lemmas 1 and 2 we have $r(d,b) \geq r(d',b)$, and by Corollary 1 there exists a minimal sequence of operations from d' to b containing only integral deletions. Thus, we can lengthen the head of the minimal sequence that does not contain nonintegral insertions. $\triangle$

All operations except for deletion and insertion will be referred to as *rejoins*.

Lemma 3. *Let, to some structure d, a rejoin o splitting a block r be applied. There exists an integral rejoin operation o' (maybe empty) for which $o'(d)$ is a simplification of $o(d)$.*

Scheme of the proof. For each operation o one can specify a "substituting" operation o' which does not split the block: the cut that splits the block in o is moved to the end of the block. $\triangle$

A detailed proof of Lemma 3 is deferred to the Appendix at `http://lab6.iitp.ru/ru/pr_chromo/`.

Corollary 3. *There exists a minimal integral sequence of operations from a to b.*

Proof. By Corollary 2, there exists a minimal sequence S in which all deletions and insertions are integral. Consider the first nonintegral rejoin operation o in it; denote by d the structure obtained by applying this operation. By Lemma 3 (taking into account Lemmas 1 and 2), there exists an integral rejoin o' such that $r(o'(d),b) \leq r(o(d),b)$. By Corollary 2, there exists a minimal sequence of operations transforming $o'(d)$ into b in which all deletions and insertions are integral. Let us transform the sequence S by removing nonintegral rejoins from it and preserving the integrality of deletions and insertions. Corollary 3 and therefore Claim 1 of Theorem 1 are proved. $\triangle$

The next lemma makes it possible to move special operations to the head of a minimal sequence.

Lemma 4. *Let d be a structure, and let $f = o_2(o_1(d))$, where o_1 is an ordinary operation, o_2 is special, and both are integral. There exist integral operations o_3 and o_4 (one of them may be empty) such that the structure $o_4(o_3(d))$ is a simplification of f and o_4 is an ordinary operation.*

Scheme of the proof. Consider arbitrary operations o_1 and o_2. Since o_1 is an ordinary operation, it preserves all blocks. First we can join the two blocks that were joined by o_2 and then "complete" what was done by o_1. $\triangle$

A detailed proof of Lemma 4 is deferred to the Appendix at `http://lab6.iitp.ru/ru/pr_chromo/`.

A minimal integral sequence of operations transforming a into b will be called *canonical* if all special operations in it precede all ordinary ones.

Corollary 4. *There exists a canonical sequence of operations transforming a into b.*

Proof. By Corollary 3 there exists a minimal integral sequence S transforming a into b. If there is an ordinary operation in it preceding a special one, we swap them by Lemma 4. After finitely many such swaps, S takes the desired form. $\triangle$

In a canonical sequence of operations, all deletions are made before all insertions, which proves Claim 2 of Theorem 1. $\triangle$

Due to Theorem 1, in what follows we may consider only five operations; in fact, they are considered over a common graph only. Thus, over a common graph there can be performed four original operations (Section 2.3) and the deletion operation. This is deletion of a special vertex (block): if it is of degree 2, it is deleted and edges incident to it merge into a single edge with the same label; if it is of degree 1, it is deleted together with the incident edge; if it is of degree 0 or has a loop, the vertex and the loop are deleted.

Corollary 5. *The problem of finding a minimal sequence transforming a into b reduces to the problem of reducing a common graph $a + b$ to a final form $c + c$ using original operations and only one additional operation of deleting a block.*

In fact, these problems are equivalent, but the above-stated one-way implication is sufficient for our purposes.

Proof. A solution of the second problem is a pair of sequences (S_1, S_2) such that the first transforms a into some structure c, and the second transforms b into the same structure c. By the solution length we call the sum of the numbers of operations in S_1 and S_2. Concatenation of S_1 and the reverse of S_2 gives a sequence of the same length transforming a into b. It remains to prove that it is minimal. To this end, it suffices to show that a solution of the second problem has length not greater than a solution of the first. By Theorem 1, the first problem has a solution satisfying properties 1 and 2 of this theorem. In the correspondent sequence S, take the structure c obtained after all deletions but before all insertions. Then all operations to the left of it do not contain insertions and do not split a-blocks. Consider the part of the sequence S between c and b written in the reverse order. This a minimal sequence transforming b into c. By Theorem 1 there exists a sequence of the same length which transforms b into c and does not contain insertions and operations that split b-blocks. Together with the initial part of S, it gives a solution of the second problem of the same length as the solution of the first. △

By a *segment* in a common graph we call a maximal fragment of ordinary edges (possibly a cycle) not contained in a 2-cycle. A segment is said to be *odd* (*even*) if the number of edges in it is odd (even). A segment is *extreme* if it contains an edge incident to no more than one edge of the graph. Note that an isolated vertex is not a segment, and an isolated ordinary edge is an extreme segment.

3.2. Algorithm for the Reduction of a Common Graph to a Special Form

Delete all special loops. For any noncyclic segment s in the graph $a + b$ do: while in s there is an edge e incident to two nonhanging edges of the graph at least one of them being ordinary, circularize e by a 2-break; there appears a cycle of two ordinary edges. Thus two adjacent edges merge into one (special if one of them was special; see Fig. 1a). This operation is ordinary and corresponds to cutting two adjacent ordinary edges from s.

As a result, the number of edges in s becomes either 1 or 2 (if s does not vanish). If this number is 2, then either s is a path of length 2 without special edges (it is circularized by a 1.5-break; see [12, Fig. 3b]) or s contains an edge located between another edge in s and a hanging edge. In the latter case it is circularized by a 2-break into a 2-cycle; then a neighboring edge becomes hanging (see Fig. 1b). This operation is ordinary and corresponds to cutting two adjacent edges from s, whereupon s vanishes.

Let the number of edges in s be 1. If a segment s is not extreme, then its unique edge is between two special edges; one or two of them can be hanging. It is circularized by a special 2-break into a 2-cycle (see Fig. 1c).

If s is an extreme segment, then its unique edge forms a path of length 1 (it is either circularized by an ordinary fusion; see [12, Fig. 3c]) or has exactly one neighboring edge (possibly, hanging). It is circularized by an ordinary 1.5-break into a 2-cycle (see Fig. 1d).

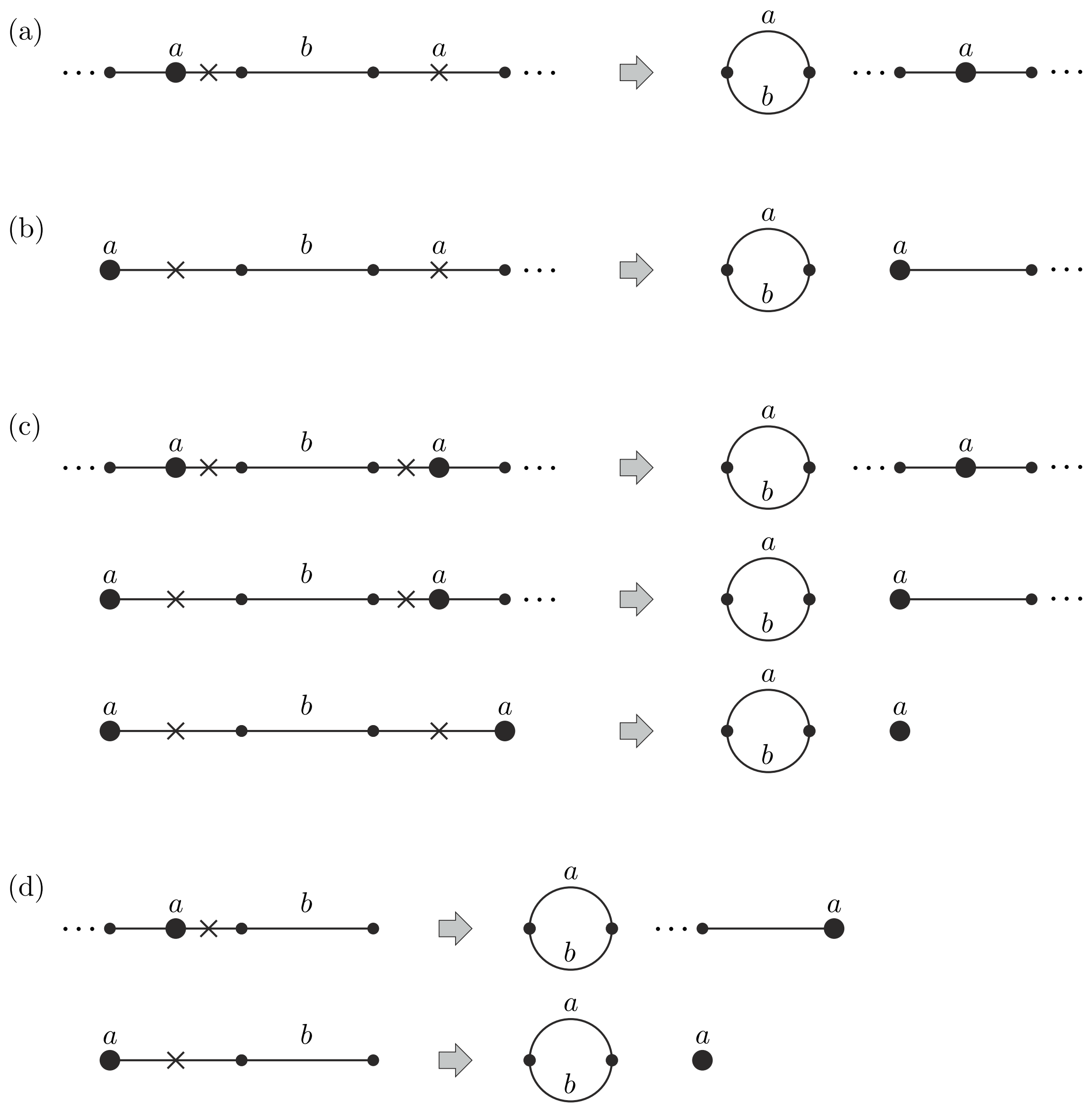

Fig. 1. To the algorithm for reducing a common graph to a special form: (a) Deleting a pair of ordinary edges (main case). Special vertices are denoted by big black circles; (b) Deleting a pair of ordinary edges (additional case); (c) Deleting an ordinary edge (three cases). In the third case two special vertices merge into one isolated special vertex; (d) Deleting an extreme ordinary edge (two cases).

Every cyclic segment is reduced to a final form by 2-break operations, which isolate 2-cycles from it (see [12, Fig. 3d]).

It is clear that as a result we obtain a graph of a special form.

By the *length* of a segment we call the number of edges in it. For each segment, the number of ordinary operations in the algorithm equals half the segment length if it is even, is less by 0.5 if it is odd and the segment is not extreme, and is greater by 0.5 if it is odd and the segment is extreme. For each cyclic segment, the number of ordinary operations is less by 1 than one half of its length.

3.3. Algorithm for the Reduction of a Common Graph to a Final Form

A graph $a + b$ is given. The following steps are executed.

Step 1. Delete all special loops (i.e., blocks nonadjacent to anything). Transform ordinary edges that do not belong to 2-cycles into 2-cycles. After that, only special edges, 2-cycles, and isolated vertices remain. The result of this step is a graph $B(a+b)$ constructed in Section 3.2. Now our goal is splitting all cycles of lengths strictly greater than 2 and paths of lengths strictly greater than 0 into 2-cycles, and deleting all special edges from them. This results in a graph of a final form.

Step 2. To each cycle and each path in $B(a+b)$ we assign its type.

An *a-path* is either an odd path whose extreme nonhanging edges are labeled with a or a path which is an isolated vertex labeled with b. A *b-path* is defined similarly.

To a cycle, we assign the type "cycle." For an a-path, its type is defined as follows: $1a$ if it has one hanging edge; $2a$ if it either has two such edges or it is an isolated vertex labeled with b; $3a$ if it has no hanging edges. For a b-path, the definition is analogous. For an even path, its type is defined as follows: 1 if it has one hanging edge; 2 if it has two such edges; 3 if it has no hanging edges; 0 if it is an isolated ordinary vertex. For even paths, there is no need for distinguishing a- and b-paths.

Step 3. Let us describe the totality of sets, each set in which is assigned with its weight, the sum of weights of all its elements. Each element is a set of paths in the graph $B(a+b)$. A set must consist of pairwise disjoint elements.

The weight of an element specifies the gain in the number of ordinary operations when applying the algorithm to the element. The algorithm uses any *set* M of the maximal weight from this totality; it will also be inclusion-maximal. It remains to determine what types of paths may occur in an element and assign a weight to it.

Thus, admissible elements are those composed of paths of the following types, one path from each type; the weight of an element is given in parentheses: $\{1a, 1b\}$ (2), $\{2a, 3b\}$ (1), $\{2b, 3a\}$ (1), $\{2, 3\}$ (1), $\{1a, 2b, 3\}$ (2), $\{1b, 2a, 3\}$ (2), $\{1a, 3b, 2\}$ (2), $\{1b, 3a, 2\}$ (2), $\{1a, 2\}$ (1), $\{1b, 2\}$ (1), $\{1a, 3\}$ (1), $\{1b, 3\}$ (1), $\{1a, 1a, 2b, 3b\}$ (3), $\{1b, 1b, 2a, 3a\}$ (3), $\{1a, 1a, 2b\}$ (2), $\{1b, 1b, 2a\}$ (2) $\{1a, 1a, 3b\}$ (2), $\{1b, 1b, 3a\}$ (2), $\{1a, 1a\}$ (1), $\{1b, 1b\}$ (1), $\{1a, 2b\}$ (1), $\{1b, 2a\}$ (1), $\{1a, 3b\}$ (1), $\{1b, 3a\}$ (1), $\{2a, 2b, 3, 3\}$ (2), $\{3a, 3b, 2, 2\}$ (2), $\{2, 2, 3a\}$ (1), $\{2, 2, 3b\}$ (1), $\{3, 3, 2a\}$ (1), $\{3, 3, 2b\}$ (1), $\{2a, 2b, 3\}$ (1), $\{3a, 3b, 2\}$ (1). The size of an element takes values from 2 to 4.

Step 4. Cycles, as well as paths that do not belong to M, are finalized separately and independently of each other. Namely, paths of length -1 are deleted. In 2-cycles and paths of length 0, special edges are deleted. Cycles of length strictly greater than 2 are split into 2-cycles, from which special edges are deleted. Paths of lengths strictly greater than 0 are circularized into cycles, which are finalized as is described above. Here an odd path is circularized by joining its ends. An even path is circularized by a 1.5-break with cutting an extreme ordinary vertex (possibly, with a hanging edge, which is deleted after that) and joining the obtained extremity with the opposite end of the path.

Step 5. For each element e in M, define a graph $G(e)$ consisting of paths contained in e and finalize it as follows. Below we consider one (the first) of the possible cases; the second case is similar.

A path is said to be *special* if it entirely consists of special edges. We apply Step 4 to the results of the following substeps. It is worth noting that in all the substeps the operation is special.

5.1. Element of the type $\{1a, 1b\}$. A 1.5-break. In the $1a$-path we make an exterior cut (i.e., cut the extreme nonhanging edge), and join the special vertex with the end of the hanging edge of the $1b$-path. This results in a path of type 0 and a special even path of type 1 (by applying Step 4 to it, we finalize it independently of other paths).

5.2. Element of the type $\{1a, 1a\}$ or $\{1b, 1b\}$. Join the ends of the hanging edges of two $1a$-paths. This results in a special odd $3a$-path.

5.3. Element of the type $\{1a, 2b\}$ or $\{1b, 2a\}$. In the $1a$-path we make an exterior cut and join the special vertex with either the end of the hanging edge of the $2b$-path or with the isolated special vertex (if the $2b$-path is such), a 1.5-break. This results in a path of type 0 and a special even path of type 2. Hereinafter, up to Step 5.8, we use a 1.5-break.

5.4. Element of the type $\{1a, 3b\}$ or $\{1b, 3a\}$. In the $3b$-path we make an exterior cut and join the special vertex with the end of the hanging edge of the $1a$-path. This results in a path of type 0 and a special even path of type 3.

5.5. Element of the type $\{1a, 2\}$ or $\{1b, 2\}$. In the $1a$-path we make an exterior cut and join the special vertex with the end of the hanging edge of the 2-path. This results in a path of type 0 and a special odd path of type $2a$.

5.6. Element of the type $\{1a, 3\}$ or $\{1b, 3\}$. In the 3-path we make a cut of the extremal b-edge and join the special vertex with the end of the hanging edge of the $1a$-path. This results in a path of type 0 and a special odd path of type $3a$.

5.7. Element of the type $\{2a, 3b\}$ or $\{2b, 3a\}$. In the $3b$-path we make an exterior cut and join the special vertex with either the end of the hanging edge of the $2a$-path of with the isolated special vertex (if the $2a$-path is such). This results in a path of type 0 and a special even path of type 1.

5.8. Element of the type $\{2, 3\}$. In the 3-path we make an exterior cut and join the special vertex with the end of the hanging edge of the 2-path, a 1.5-break. This results in a path of type 0 and a special even path of type 1.

5.9. Element of the type $\{1a, 1a, 2b\}$ or $\{1b, 1b, 2a\}$. To the two $1a$-paths we apply Step 5.2. This results in a special odd path of type $3a$. To it and the initial $2b$-path, we apply Step 5.7.

5.10. Element of the type $\{1a, 1a, 3b\}$ or $\{1b, 1b, 3a\}$. To the paths $1a$ and $3b$ we apply Step 5.4. This results in a special even path of type 3 and a path of type 0. To the first of them and the second initial $1a$-path, we apply Step 5.6.

5.11. Element of the type $\{1a, 2b, 3\}$ or $\{1b, 2a, 3\}$. To the paths $1a$ and $2b$ we apply Step 5.3. This results in a special even path of type 2 and a path of type 0. To the first of them and the initial 3-path, we apply Step 5.8.

5.12. Element of the type $\{1a, 3b, 2\}$ or $\{1b, 3a, 2\}$. To the paths $1a$ and $3b$ we apply Step 5.4. This results in a special even path of type 3 and a path of type 0. To the first of them and the initial 2-path, we apply Step 5.8.

5.13. Element of the type $\{2a, 2b, 3\}$. In the 3-path we cut the exterior b-edge and join the special vertex with either the end of the hanging edge of the $2a$-path or the isolated special vertex (if the $2a$-path is such), a 1.5-break. This results in a special odd path of type $1a$ and a path of type 0. Then to the pair of paths $\{1a, 2b\}$ we apply Step 5.3.

5.14. Element of the type $\{3a, 3b, 2\}$. In the $3a$-path we make an exterior cut and join the special vertex with the end of the hanging edge of the 2-path, a 1.5-break. This results in a special odd path of type $1a$ and a path of type 0. Then to the pair of paths $\{1a, 3b\}$ we apply Step 5.4.

5.15. Element of the type $\{2a, 3, 3\}$ or $\{2b, 3, 3\}$. To the pair of paths $\{2a, 3\}$ we apply the first operation of Step 5.13. This results in a special odd path of type $1a$ and a path of type 0. To the pair of paths $\{1a, 3\}$ we apply Step 5.6.

5.16. Element of the type $\{3a, 2, 2\}$ or $\{3b, 2, 2\}$. To the pair of paths $\{3a, 2\}$ we apply the first operation of Step 5.14. This results in a special odd path of type $1a$ and a path of type 0. To the pair of paths $\{1a, 2\}$ we apply Step 5.5.

5.17. Element of the type $\{2a, 2b, 3, 3\}$. To the triple of paths $\{2a, 2b, 3\}$ we apply Step 5.13. This results in a special even path of type 2 and a path of type 0. To the obtained 2-path and the second initial 3-path, we apply Step 5.8.

5.18. Element of the type $\{3a, 3b, 2, 2\}$. To the triple of paths $\{3a, 3b, 2\}$ we apply Step 5.14. This results in a special even path of type 3 and a path of type 0. To the obtained 3-path and the second initial 2-path, we apply Step 5.8.

5.19. Element of the type $\{1a, 1a, 2b, 3b\}$ or $\{1b, 1b, 2a, 3a\}$. To the pair of paths $\{1a, 2b\}$ we apply Step 5.3. This results in a special even path of type 2 and a path of type 0. To the other pair of initial paths $\{1a, 3b\}$ we apply Step 5.4. This results in a special even path of type 3 and a path of type 0. To the obtained 2-path and 3-path, we apply Step 5.8.

3.4. Correctness and Linear Complexity of the Algorithm for the Reduction of a Common Graph to a Final Form

1. From the description of the algorithm it is seen that it reduces the graph $G = a + b$ to a final form. Let us estimate the runtime of the algorithm. Let the number of vertices in G be n. Then deleting special loops requires no more than n operations; deleting ordinary edges also requires at most n operations. When constructing a set M, we add one by one no more than n sets of paths of sizes from 2 to 4; when finalizing components that do not belong to M, each component of size m requires at most one operation of circularizing a path, at most m operation of cutting 2-cycles from a cycle, and at most m operations of deleting a block; when finalizing sets of paths, each set of size m requires at most three operations of interaction between the components, at most one operation of circularizing a path, at most m operations of cutting 2-cycles from a cycle, and at most m operations of block deletion. Thus, the algorithm runtime is linear in n.

By the definition of a sequence reducing $a + b$ to a final form, the number of special operations equals B, where B is the number of special vertices (blocks) in the graph G.

2. Define the *defect* of a component of the graph $B(a + b)$ to be

- 0 if it is a cycle or a path of type $2a$, $2b$, 0, 1, or 2;
- 1 if it is a path of type $1a$, $1b$, $3a$, $3b$, or 3.

Intuitively, the defect is the number of ordinary operations in the minimal sequence for this component.

Let D be the sum of defects of all components of the graph $B(a + b)$, P be the weight of M, and S be the sum of integer parts of halved segment lengths in the graph $a + b$ plus the number of odd extreme segments minus the number of cyclic segments. In the next subsection, we present an algorithm for constructing a set M.

The number of ordinary operations in a sequence constructed by the algorithm is $S + D - P$ (Lemma 5).

3. Introduce the notation $t(G) = B + S + D - P$. Any operation changes the value of $t(G)$ by at most 1 (Theorem 5).

4. Any sequence of operations of length strictly less than $t(G)$ does not reduce the graph G to a final form (Theorem 6).

Items 2–4 imply the correctness of the algorithm; i.e., it produces a minimal sequence. Claims given in these items are proved in Section 3.6.

3.5. Algorithm for Constructing a Set of Elements of the Maximum Weight

In the general case, finding a set of maximum cardinality in a given collection of sets is an NP-hard problem. In our case this is overcome due to a special order in our collection of sets. As a result, we get a linear complexity algorithm.

We will refer to types 1, 2, and 3 as even, and to types $1a$, $1b$, $2a$, $2b$, $3a$, and $3b$ as odd. Types 0 and "cycle" do not enter the elements. In the proof, we essentially use the following two bijections on the set of path types. The first of them maps members of the pairs $(1a, 1b)$, $(2a, 2b)$, and $(3a, 3b)$ into each other and fixes even types; we call it the *horizontal automorphism*. The second bijection maps members of the pairs $(2a, 3a)$, $(2b, 3b)$, and $(2, 3)$ into each other and fixes the other types; we call it the *vertical automorphism*. Both bijections "preserve admissibility of the type" of an element in the sense that any automorphism maps an admissible type to an admissible type.

A set M of maximum-weight elements is constructed by successively adding elements to it. Initially, M is empty. If T is the type of an element, we define the *increment* of M by T to be adding to M the maximal number of elements of type T disjunctive with each other and with M; a distribution of paths among these elements can be arbitrary. This number equals the minimum (over types t in T) cardinality of a set of paths of type t that are not contained in a current M. A path which is not contained in M will be called *free*.

The algorithm consists of the following successive increments.

By type $\{1a, 1b\}$, after which there are either no free paths of type $1a$ or no free paths of type $1b$. We consider the first case; otherwise, in all subsequent steps a should be changed to b.

Then by types $\{2a, 3b\}$ and $\{2b, 3a\}$; by type $\{2, 3\}$; by types $\{1a, 2b, 3\}$ and $\{1a, 3b, 2\}$; by types $\{1a, 2\}$ and $\{1a, 3\}$; by type $\{1a, 1a, 2b, 3b\}$; by types $\{1a, 1a, 2b\}$ and $\{1a, 1a, 3b\}$; by type $\{1a, 1a\}$; by types $\{1a, 2b\}$ and $\{1a, 3b\}$; by types $\{2a, 2b, 3, 3\}$ and $\{3a, 3b, 2, 2\}$; by types $\{2, 2, 3a\}$ and $\{3, 3, 2a\}$; by types $\{2, 2, 3b\}$ and $\{3, 3, 2b\}$; by types $\{2a, 2b, 3\}$ and $\{3a, 3b, 2\}$.

The algorithm does not require specifying all admissible elements, it suffices to group paths in $B(a+b)$ according to their types.

Theorem 2. *The algorithm constructs a set M of the maximum weight.*

Scheme of the proof. We assume that the algorithm constructs a set M by adding elements to it one by one according to the order described above. It suffices to show that at each step a current set M is a subset of some maximum-weight set (call it a maximum set). We proceed by induction on the construction of M.

Let at a current step an element e be added to M. Let there be some maximal set M' that does not contain e but contains all elements of M. Let us show that it is possible to reorganize the set M' so that it will remain to be maximal and contain e. Consider the set K of elements of M' containing at least one path from e. Denote the set of paths contained in elements of K by $I(K)$, and the set of paths contained in e, by $I(e)$. Looking over possible combinations of types of elements of K, we make sure that, in all the cases, from paths of the set $I(K) \setminus I(e)$ it is possible to compose a set K' of pairwise disjoint elements of total weight not less than $w(K) - w(e)$, where $w(K)$ is the total weight of elements of K and $w(e)$ is the weight of e. Replacing in M' the set K with the union of K' and e, we obtain the desired maximal set. Note that a certain order of adding elements to M is essential: when looking over possible types in K, we use the fact that from paths in the union of $I(K)$ and $I(e)$ one cannot compose any element of a type that was added to M earlier (in particular, K contains no element of such a type). $\triangle$

Technical details of the proof of Theorem 2 are deferred to the Appendix at `http://lab6.iitp.ru/ru/pr_chromo/`.

3.6. Formulation of Auxiliary Statements from Section 3.4 *and Their Proofs*

Lemma 5. *The algorithm for finalizing a graph G constructs a sequence of $t(G)$ operations.*

Proof. It is clear that the number of special operations in the algorithm equals B; it remains to count the number of ordinary operations. It is seen from the description of reduction of a common graph $a + b$ to a final form that the number of ordinary operations in it equals S. Finalizing all components of $B(a+b)$ separately requires D ordinary operations (this follows from the description in Section 3.3, Step 4; see also Theorem 3). Using "reductions" with the help of sets of minimum weight reduces this estimate by P (this follows from the description in Section 3.3, Step 5; see also Theorem 4). $\triangle$

Theorem 3. *If a special common graph has a single component C, then $D(C) \geq p$, where p is the number of ordinary operations in a minimal sequence.*

Proof. The inequality $D(C) \leq p$ is also valid (see Corollary 6 below), but we do not use it. Consider all kinds of a graph C. If it is a cycle, we proceed by induction on its length. For length 2, no ordinary operations are needed. For an arbitrary length, take two (special) edges with the same label separated by precisely one edge. By a 2-break with deleting these edges we split the cycle into two cycles so that one cycle is of length 2 and the other is special (see Fig. 2a).

This operation is special, two blocks are merged into one. By the induction hypothesis, a special cycle of smaller length can be finalized by a sequence of special operations. Then C is also finalized by such a sequence, as required. Consider the case of a linear component.

Case of odd paths (see Fig. 2b).

Type 1. By joining the end of a hanging edge with the opposite end of the path, we obtain a special cycle (ordinary operation), the top arrow in Fig. 2b. By what was proved above, a special cycle can be finalized by a sequence of special operations. Then C can be finalized by a sequence consisting of special operations and one ordinary operation, as required.

Type 2. If C is a path of length -1, no ordinary operations are needed. Otherwise, by joining the ends of hanging edges, we obtain a special cycle (special operation), the bottom arrow in Fig. 2b. Further reasoning is similar to the above.

Type 3. By joining the ends of the path by an edge, we obtain a cycle with one ordinary edge (such a cycle will be called *semispecial*), the middle arrow in Fig. 2b; the upper block of the cycle is regarded as absent. This is an ordinary operation. If the cycle is of length 2, there is nothing to prove. Otherwise, by deleting an ordinary edge from the cycle (as in the algorithm for reducing a graph to a special form), we obtain that a semispecial cycle can be finalized by a sequence of special operations. Then the graph C is finalized by a sequence consisting of special operations and one ordinary operations, as required.

Case of even paths (see Fig. 2c).

Type 0. The claim is obvious.

Type 1. If the path length is 0, the claim is obvious. Otherwise, we make a special 1.5-break by cutting an extreme nonhanging edge (we call this an *exterior* cut) and joining the end of a hanging edge with the extreme special vertex, the top arrow in Fig. 2c. We obtain a special cycle and a path of type 0. Thus, the graph C is finalized by a sequence of special operations, as required.

Type 2. If the path length is 0, the claim is obvious. Otherwise, we make a special 1.5-break with an exterior cut of an extreme nonhanging edge and join of an extreme special vertex with the end of the opposite hanging edge, the bottom arrow in Fig. 2c. This results in a special cycle and a path of length 0 with one hanging edge. Further reasoning is obvious.

Type 3. Make a 1.5-break with an exterior cut and join of an extreme special vertex with the opposite end of the path, the middle arrow in Fig. 2c. This is an ordinary operation, which results in a special cycle and a path of type 0. $\triangle$

For any element e, denote by $G(e)$ the graph consisting of its paths; by $D(e)$, the sum of defects of all paths in e; and by $c(e)$, the weight of e.

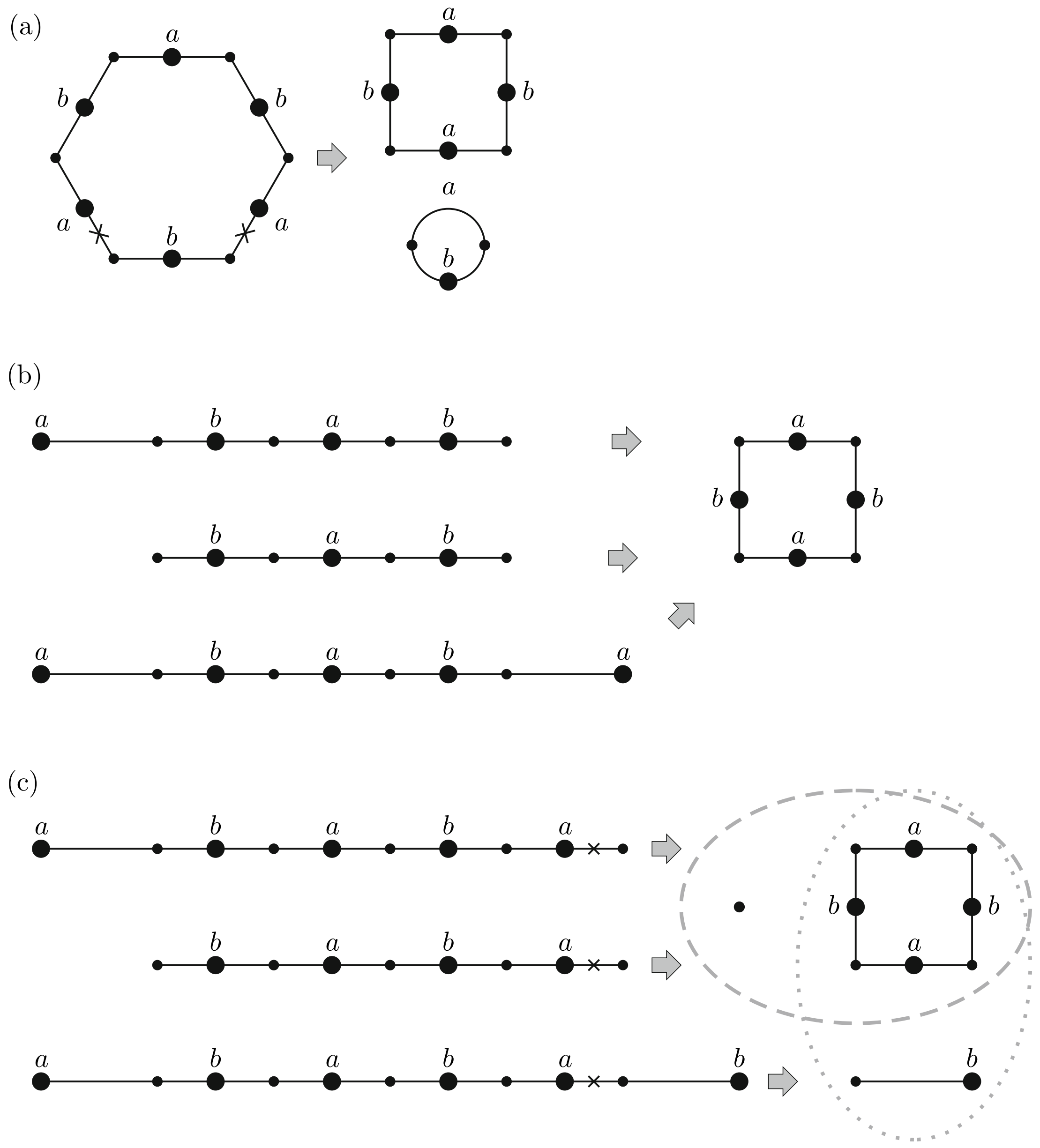

Fig. 2. To the proof of Theorem 3: (a) Special operation of splitting a cycle into two; (b) Circularization of odd paths (top and middle: ordinary operations; bottom: special operation); (c) Circularizaton of even paths (top and bottom: special operations; middle: ordinary operation).

Theorem 4. *The number p of ordinary operations in a minimal sequence for $G(e)$ is not greater than $D(e) - c(e)$.*

Proof. One can easily check that in each of Cases 5.1–5.19 described in Section 3.3, for joining the paths in an element e into a single path, only special operations are used and that the defect of the resulting path equals $D(e) - c(e)$. By Theorem 3, this path is finalized using at most $D(e) - c(e)$ ordinary operations. Thus, finalizing the graph $G(e)$ requires at most $D(e) - c(e)$ ordinary operations. To simplify this check, we present results of each case as an equality: the left-hand side is the sum of initial defects of a path, the right-hand side is the type of the resulting path, and in the square brackets we give the difference of the sum of defects of the left-hand paths and the defect of the right-hand path, which equals $c(e)$. Then

5.1. $1a + 1b = 1$ [2];
5.2. $1a + 1a = 3a$ [1];
5.3. $1a + 2b = 2$ [1];
5.4. $1a + 3b = 3$ [1];
5.5. $1a + 2 = 2a$ [1];
5.6. $1a + 3 = 3a$ [1];
5.7. $2a + 3b = 1$ [1];
5.8. $2 + 3 = 1$ [1];
5.9. $1a + 1a + 2b = 3a + 2b = 1$ [2];
5.10. $1a + 1a + 3b = 1a + 3 = 3a$ [2];
5.11. $1a + 2b + 3 = 2 + 3 = 1$ [2];
5.12. $1a + 3b + 2 = 3 + 2 = 1$ [2];
5.13. $2a + 2b + 3 = 2a + 1b = 2$ [2];
5.14. $3a + 3b + 2 = 3a + 1b = 3$ [1];
5.15. $2a + 3 + 3 = 1a + 3 = 3a$ [1];
5.16. $3a + 2 + 2 = 1a + 2 = 2a$ [1];
5.17. $2a + 2b + 3 + 3 = 1a + 1b = 1$ [2];
5.18. $3a + 3b + 2 + 2 = 1a + 1b = 1$ [2];
5.19. $1a + 1a + 2b + 3b = 2 + 3 = 1$ [3]. $\triangle$

The inequality $D(e) - c(e) \leq p$ is also valid (see Corollary 6 below), and therefore Theorem 4 holds with equality.

Theorem 5. *For any operation o and any common graph G we have the inequality $t(G) - 1 \leq t(o(G)) \leq t(G) + 1$.*

Scheme of the proof. A component of a common graph will be called *standard* if it has no special edges. For a connected nonstandard component C of a graph $a + b$ we denote by $B'(C)$ a nonfinal component of the graph $B(C)$ (clearly, it is unique). The following lemma allows to quickly determine for each component C of the original graph $a + b$ the type of the corresponding component in $B(a + b)$. Thus, the classification of components of a special common graph by their types is extended to components of an arbitrary common graph.

Lemma 6. *The following statements hold true.*

1. *A graph $B(C)$ is of a final form if and only if the component C is standard*;
2. *$B'(C)$ is a special cycle if and only if C is a nonstandard cycle*;
3. *$B'(C)$ is a path of type* 1a (*similarly for* 1b) *if and only if C is a nonstandard odd a-path ending on one side with an extreme even segment* (*or this segment is absent; then we assume it to be an even segment of length* 0) *and on the other side with either a hanging edge or an extreme odd segment*;
4. *$B'(C)$ is a path of type* 2a (*similarly for* 2b) *if and only if C is a nonstandard odd a-path ending on both sides with either a hanging edge or an extreme odd segment* (*this type includes also an isolated special vertex labeled with b*);
5. *$B'(C)$ is a path of type* 3a (*similarly for* 3b) *if and only if C is a nonstandard odd a-path ending on both sides with an extreme even segment*;
6. *$B'(C)$ is a path of type* 1 *if and only if C is a nonstandard even path ending on one side with an extreme even segment and on the other side with either a hanging edge or an extreme odd segment*;
7. *$B'(C)$ is a path of type* 2 *if and only if C is a nonstandard even path ending on both sides with either a hanging edge or an extreme odd segment*;
8. *$B'(C)$ is a path of type* 3 *if and only if C is a nonstandard even path ending on both sides with an extreme even segment.*

Proof. We say that a component C is of kind i, $i = 1, 2, \ldots, 8$, if it corresponds to the description on the right-hand side of the equivalence proved in item i of Lemma 6. Clearly, each component has a unique kind. Therefore, it suffices to prove all the equivalences from the right to the left. For the proof, it suffices to check that each operation in the algorithm of reducing a graph to a special form does not change the kind of a component to which it is applied, which is obvious. △

Further proof of Theorem 5 consists, in essence, in examining all possible types of an operation o and its arguments. For all operations except for a 2-break the examination is straightforward. For instance, let o be the operation of deleting an isolated special vertex (recall that it is considered to be a path of type $2a$ or $2b$). Then, when passing from a graph G to $o(G)$, the value of B reduces by 1 and the values of S and D do not change. The value of P is either unchanged or reduced by 1. Indeed, it cannot reduce by more than 1, since any element of weight 2 (respectively, 3) after deleting from it a path of type $2a$ or $2b$ transforms into an element of weight 1 (respectively, 2). Thus, the value of $t(G)$ is either kept unchanged or reduces by 1, which implies the desired statement.

The case of a 2-break applied to either a single component or two components which are a path and a cycle is considered similarly. In the case where a 2-break o is applied to two path, a direct search of possibilities is difficult because there are too many of them. More precisely, difficult is the case where there are no blocks on one side of some cut in o. Then the question reduces to the case of a 1.5-break or a fusion, as is described in item 5.3.2 of the proof of Theorem 5 at `http://lab6.iitp.ru/ru/pr_chromo/`, where this proof is given in full detail. △

Theorem 6. *The algorithm for reducing a graph $a + b$ to a final form constructs a minimal sequence of operations. The algorithm runtime is linear in the size of the common graph of the two initial structures.*

Proof. For a final graph we have $t(G) = 0$. Reducing the initial value of $t(G)$ to 0 in less than $t(G)$ operations is possible only if some operation reduces $t(G)$ by more than 1. This contradicts Theorem 5. The linearity of the algorithm runtime follows from its description. △

Corollary 6. *In the notation of Theorems* 3 *and* 4 *we have the estimates $D(C) \leq p$ and $D(e) - c(e) \leq p$.*

Proof. Clearly, $t(C) = B + D(C)$, where B is the number of blocks in C. If the component C could be finalized using less than $D(C)$ ordinary operations, then in total we would use less than $t(C)$ operations. This contradicts Theorem 6 taking into account Lemma 5. The proof of the second inequality is similar, taking into account the obvious equality $t(G(e)) = B + D(e) - c(e)$, where B is the number of blocks in $G(e)$. △

The authors are grateful to a reviewer for valuable remarks.

REFERENCES

1. Blanchette, M., Kunisawa, T., and Sankoff, D., Gene Order Breakpoint Evidence in Animal Mitochondrial Phylogeny, *J. Mol. Evol.*, 1999, vol. 49, no. 2, pp. 193–203.
2. Lyubetsky, V.A., Gershgorin, R.A., Seliverstov, A.V., and Gorbunov, K.Yu., Algorithms for Reconstruction of Chromosomal Structures, *BMC Bioinformatics*, 2016, vol. 17, Research Article 40.
3. Yancopoulos, S., Attie, O., and Friedberg, R., Efficient Sorting of Genomic Permutations by Translocation, Inversion and Block Interchange, *Bioinformatics*, 2005, vol. 21, no. 16, pp. 3340–3346.
4. Hannenhalli, S. and Pevzner, P.A., Transforming Men into Mice (Polynomial Algorithm for Genomic Distance Problem), in *Proc. 36th Annual Sympos. on Foundations of Computer Science (FOCS'95), Milwaukee, WI, USA, Oct. 23–25, 1995*, pp. 581–592.
5. Bergeron, A., Mixtacki, J., and Stoye, J., A Unifying View of Genome Rearrangements, *Algorithms in Bioinformatics (Proc. 6th Int. Workshop, WABI'2006, Zurich, Switzerland, Sept. 11–13, 2006)*,

Bucher, P. and Moret, B.M.E., Eds., Lect. Notes Comp. Sci., vol. 4175, Berlin: Springer, 2006, pp. 163–173.

6. Braga, M.D.V., Willing, E., and Stoye, J., Genomic Distance with DCJ and Indels, *Algorithms in Bioinformatics (Proc. 10th Int. Workshop, WABI'2010, Liverpool, UK, Sept. 6–8, 2010)*, Moulton, V. and Singh, M., Eds., Lect. Notes Bioinformat., vol. 6293, Berlin: Springer, 2010, pp. 90–101.

7. Braga, M.D.V., Willing, E., and Stoye, J., Double Cut and Join with Insertions and Deletions, *J. Comput. Biol.*, 2011, vol. 18, no. 9, pp. 1167–1184.

8. da Silva, P.H., Braga, M.D.V., Machado, R., and Dantas, S., DCJ-indel Distance with Distinct Operation Costs, *Algorithms in Bioinformatics (Proc. 12th Int. Workshop, WABI'2012, Ljubljana, Slovenia, Sept. 10–12, 2012)*, Raphael, B.J. and Tang, J., Eds., Lect. Notes Bioinformat., vol. 7534, Berlin: Springer, 2012, pp. 378–390.

9. Compeau, P.E.C., A Simplified View of DCJ-Indel Distance, *Algorithms in Bioinformatics (Proc. 12th Int. Workshop, WABI'2012, Ljubljana, Slovenia, Sept. 10–12, 2012)*, Raphael, B.J. and Tang, J., Eds., Lect. Notes Bioinformat., vol. 7534, Berlin: Springer, 2012, pp. 365–377.

10. Compeau, P.E.C., DCJ-Indel Sorting Revisited, *Algorithms Mol. Biol.*, 2013, vol. 8, Research Article 6.

11. Compeau, P.E.C., A Generalized Cost Model for DCJ-Indel Sorting, *Algorithms in Bioinformatics (Proc. 14th Int. Workshop, WABI'2014, Wrocław, Poland, Sept. 8–10, 2014)*, Brown, D.G. and Morgenstern, B., Eds., Lect. Notes Bioinformat., vol. 8701, Berlin: Springer, 2014, pp. 38–51.

12. Gorbunov, K.Yu., Gershgorin, R.A., and Lyubetsky, V.A., Rearrangement and Inference of Chromosome Structures, *Molekulyarnaya Biologiya*, 2015, vol. 49, no. 3, pp. 372–383 [*Mol. Biol.* (Engl. Transl.), 2015, vol. 49, no. 3, pp. 327–338].

13. Alekseyev, M.A. and Pevzner, P.A., Multi-Break Rearrangements and Chromosomal Evolution, *Theor. Comput. Sci.*, 2008, vol. 395, no. 2–3, pp. 193–202.

14. Alekseyev, M.A., Multi-Break Rearrangements and Breakpoint Re-uses: From Circular to Linear Genomes, *J. Comput. Biol.*, 2008, vol. 15, no. 8, pp. 1117–1131.

15. Gershgorin, R.A., Gorbunov, K.Yu., Seliverstov, A.V., and Lyubetsky, V.A., Evolution of Chromosome Structures, in *Proc. 39th IITP RAS Interdisciplinary Conference & School on Information Technology and Systems 2015 (ITaS'2015), Sochi, Russia, Sept. 7–11, 2015*, pp. 105–120.

16. Gorbunov, K.Yu. and Lyubetsky, V.A., Problems and Algorithms Related to Chromosome Rearrangements, *Sovremennye Informatsionnye Tekhnologii i IT-obrazovanie*, 2013, vol. 9, pp. 764–768.

17. Lyubetsky, V.A. and Gorbunov, K.Yu., Chromosome Structures Reconstruction, *Molecular Phylogenetics: Contributions to the 4th Moscow Int. Conf. on Molecular Phylogenetics (MolPhy-4), Moscow, Russia, Sept. 23–26, 2014*, Moscow: Torus-Press, 2014, p. 42.

18. Lyubetsky, V.A., Seliverstov, A.V., and Gorbunov, K.Yu., Rearrangement of Chromosomes: Problems, Algorithms, Databases, and Gene Expression Regulations, The 9th Int. Conf. on Bioinformatics of Genome Regulation and Structure\Systems Biology (BGRS\SB'2014), Novosibirsk, Russia, June 23–28, 2014, oral presentation.