ChromoGGL user manual

(authors: K.Yu. Gorbunov, V.A. Lyubetsky,

developers: R.A. Gershgorin, K.Yu. Gorbunov)

2019

Contents

Overview 2

The algorithm for computing the distance between two chromosome structures 2

The algorithm for transforming a joint graph into the final form in case of different operation weights and extra operations of deletion and insertion 4

Algorithm of generation of the evolutionary tree of chromosome structures .... **Error! Bookmark not defined.**

Algorithm “gradient descent” from multiple random initial points for task 3 ..... **Error! Bookmark not defined.**

Installation and execution of ChromoGGL utilities on Windows 14

Chromo ..... **Error! Bookmark not defined.**

hrom_reconstruction ..... **Error! Bookmark not defined.**

Input data 14

Chromo input data ..... **Error! Bookmark not defined.**

hrom_reconstruction input data ..... **Error! Bookmark not defined.**

ChromoGGL parameters 16

Chromo parameters 16

Command line parameters 16

File with operations costs 15

hrom_reconstruction parameters ..... **Error! Bookmark not defined.**

Output data 16

Examples of task 1 solutions 18

Exampels of task 2 solutions 19

hrom reconstruction output ..... **Error! Bookmark not defined.**

Biological examples for task 3 ..... **Error! Bookmark not defined.**

Recommended standart programs 19

Reference 19

Overview

ChromoGGL utilities are designed for solving the following **three** tasks:

1) Computation of the distance between the two given chromosome structures and finding a sequence of operations with the minimum total weight (distance) that transforms one structure into another. The most common definition (according to available papers) is considered as an arbitrary set of paths and cycles, representing the linear and circular chromosomes, as well as the definition of the set of allowed operations. The structures include gene paralogs, the sequence of operations permit alternate gene composition. Arbitrary weights of single operations are allowed. The task is to minimize the total weight of all operations from arbitrary sequence. The sequence, for which the minimum is reached, is called the *shortest*. The mean of the total weight of the sequence from one structure to another and vice versa is called the *distance* between them. Despite such a common task statement, our original algorithm has *linear* time and memory complexity; it has proved to be quick when performing computations on supercomputer

2) Computation of the matrix of pairwise distances for the given set of structures and generation of the tree, that matches the matrix best. These chromosome structures correspond to the leaves of the obtained tree. The algorithm with linear complexity, based on the algorithm for the first task, is used.

Next section contains the description of the task 1. This algorithm is the key element of the solution of task 2.

The algorithm for computing the distance between two chromosome structures

The model of chromosome structure is described as the finite number of the oriented chains and cycles, including loops. This set can be thought as the oriented graph. We call this graph as *chromosome structure.* The edge of the graph is defined as the *gene,* each single chain or cycle defined as *chromosome* or *component.* We attach the name to each gene, usually it is the *number i* of this gene, the number can repeat (in case of paralogs), in this case this number becomes *i,j.* Such a model usually does not include the lengths of genes and intergenic regions, as well as the content of genes and intergenic regions. The direction of gene indicates to which chain this gene belongs. The vertex of this graph defines the “place” of connection of the neighbouring genes. Usually there are lots of chains and cycles in the structure, this leads to

interaction of these components. That is why the case of multiple chromosomes differs from the case of the single chromosome.

The model includes the following *standart* operations for chromosome structures. The *double-cut-and-paste* – the cut of two pairs of ends of genes and new reconnection of these ends; *sesqui-cut-and-paste* – the cut of the pair of ends and the connection of one end with some new free end, another end stays free; *cut* of two connected ends; *join* of two free ends. If we have two chromosome structures *a* and *b*, and the gene is present in both structures, we will define this gene as the *common* gene (otherwise *special*). If the gene is present in the structure *a* and is absent in the structure *b*, we will define this gene as *a*-gene. Otherwise we define the gene as *b*-gene. The model includes two *additional* operations: the *deletion* of the region of special *a*-genes and the insertion of the region of *b*-genes.

When removing the special genes, we join the ends of common genes, which are neighbors of the removed region. Otherwise, if we insert the region of special genes inside of chromosome, we firstly cut two ends. We do not consider the cuts of special genes regions, because such operation do not improve the solution of the task.

So, we have six operations and the positive rational number is defined for each operation. We define this numbers as *costs* of the operations. Including the costs of operations into the model is important difference of this model from other models.

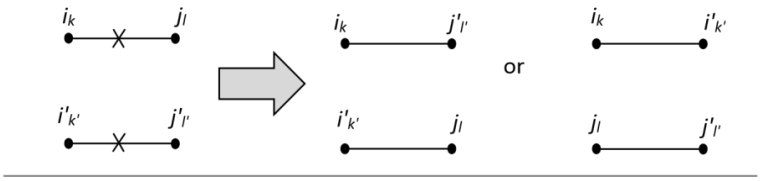
The task is to find the *shortest* sequence of this operations; which transforms the structure *a* into the structure *b*. By “shortest” sequence we mean the sequence with the minimal total cost.

The definition of the joint graph and its final form. The vertexes of the joint graph *a*+*b* are the ends of common genes and all maximal regions of the special genes. Each end of gene is included only once. We write the name of the gene with the index 1 or 2, which indicates the the beginning of the gene or its end. The vertexes of the first type are defined as the *regular* vertexes, the vertexes of the second type – *special* vertexes. We define the border edge with the special end as the *hanging* edge. Edges are marked as *a* or *b* according to the structure, in which this vertexes are connected. Two vertexes can be connected by up to two edges. The graph may contain isolated vertexes – the regions of the special genes: if we have the cyclic region, we call it the *special* loop. We now have the undirected graph.

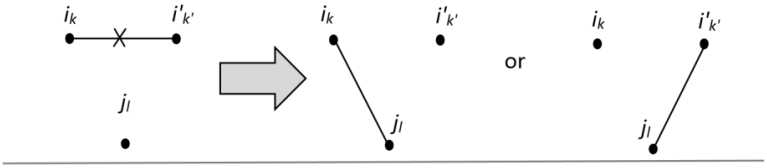
We apply the following analogs of defined operations to the joint graph.

(1) Delete two equally marked edges and reconnect free resulting vertexes by two new not incident edges. (2) remove the edge with some mark (say *a*) and connect one of resulting vertexes with another vertex, which is not incident to *a*-edge. (3) Delete any edge. (4). Connect two vertexes, not incident to *a*-edge by the *b*-edge (and vise versa). (5) Remove special vertex of special loop. If special vertex has two regular neighboring vertexes, they are being connected by the edge.

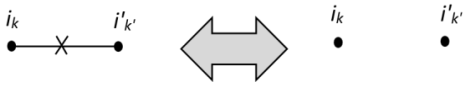
1) Double-cut-and-paste (DP). Initial edges belong to one structure



2) Sesqui-cut-and-paste (SP)



3) *a*-edge deletion or *b*-edge insertion (C) and *b*-edge deletion or *a*-edge insertion (J)

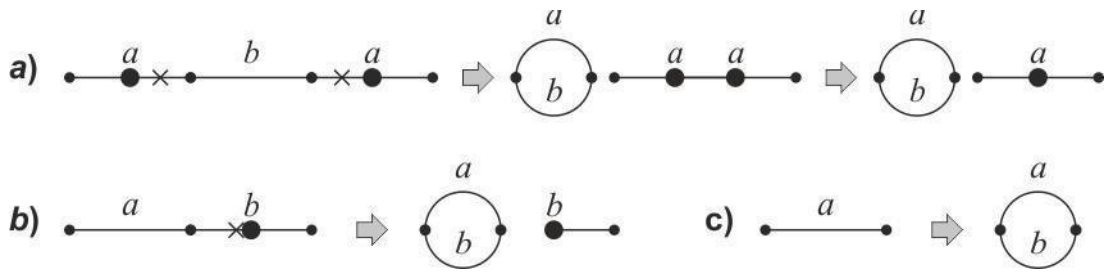


The final form of the joint graph *a*+*b* is defined as the joint graph that contains only isolated regular vertexes and *final 2-cycles.* It is easy to prove, that the *initial problem is equal* to the problem of transforming the graph *a*+*b* to the final form.

The algorithm for transforming a joint graph into the final form in case of different operation weights and extra operations of deletion and insertion

Step 1. Delete all special *a*-loops.

Step 2. Cut out a common edge not included in a 2-circle and close it into the final 2circle using a double- (internal edge) or a sesqui-cut-and-paste (extreme edge) or a join (single edge) operation. Repeat the operation if possible. If the double-cut-and-paste weight does not exceed that of sesqui-cut-and-paste, all double operations are performed first; otherwise, all sesqui operations go first.



Step 3. Let us start with definitions. An *odd* (*even*) *path* is a path of an odd (even) length. An *a-path* is an odd path with extreme non-hanging edges labeled as *a*. *b-Path* is defined in a similar way. *Types* are assined to *paths or circles remaining after steps 1-2* (except the final 2circles and isolated common nodes): 2-circles containing an *a*-node but no *b*-nodes are considered as *a-circles*; opposite structures, as *b-circles*. A circle containing both *a*- and *b*-nodes belongs to the *circle* type. Special *b*-loops belong to the *loop* type.

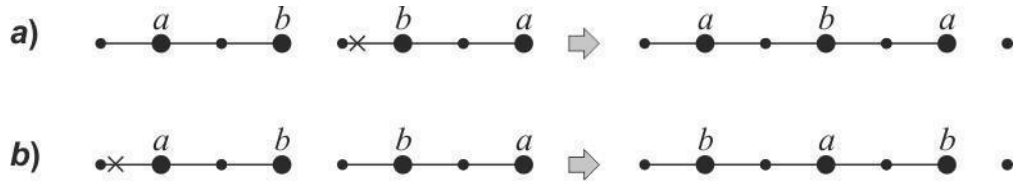
a-Paths are assigned to the following types: *1a* if the path has a single hanging edge; *2a*, if it has two hanging edges; 2*a'* -- it is an isolated special *b*-node; *3a*, if it has no hanging edges but has both *a*- and *b*-nodes (then its length is greater than 1); and *3a'*, if there are neither hanging edges nor *b*-nodes (then its length equals 1). *b*-Path types are defined in a similar way. Note, that we split paths into types “with accent” and “without accent”, because we need to mark out paths without *b*-nodes or *a*-nodes.

Even paths are assigned to the following types: *1*, if the path has a single hanging edge and a *b*node; *1'*, if it has one common node and one special *a*-node incident to it; *1''*, if it has one common node and one special *b*-node incident to it; *2*, if it has two hanging edges and non hanging edges; *2'*, if it has only two hanging edges; *3*, if it has at least one edge and no hanging edges. Type *1* is subdivided into types 1_a and 1_b if the path includes a hanging *a*-node and *b*-node, respectively.

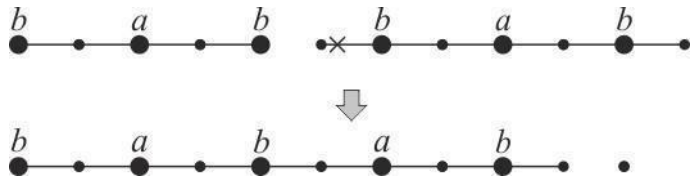
The algorithm performs the actions described below; each action is applied to a set of paths (from 2 to 4) of the corresponding types (specified in the beginning of the paragraph and separated by the plus sign. Each action is repeated as long as it is applicable. For brevity we denote 2*a* и 2*a'* as 2*a*, 3*b* and 3*b'* as 3*b*, 1_b and 1′′ as 1_b, 2 и 2′ as 2.

3.1. $1a+1b=1_c$. Cut an extreme non-hanging edge (such edges are called *external edges*) *in one of two paths* of types 1*a* and 1*b* and join the corresponding special node with the extreme special node of the other path (sesqui-cut-and-paste operation). The two variants (here, paths) are selected only at steps 4.15-4.23 of the algorithm. This uncertainty is a characteristic of our algorithm. Specifically, an intermediate path type 1_c corresponding to 1_a or 1_b is introduced. At

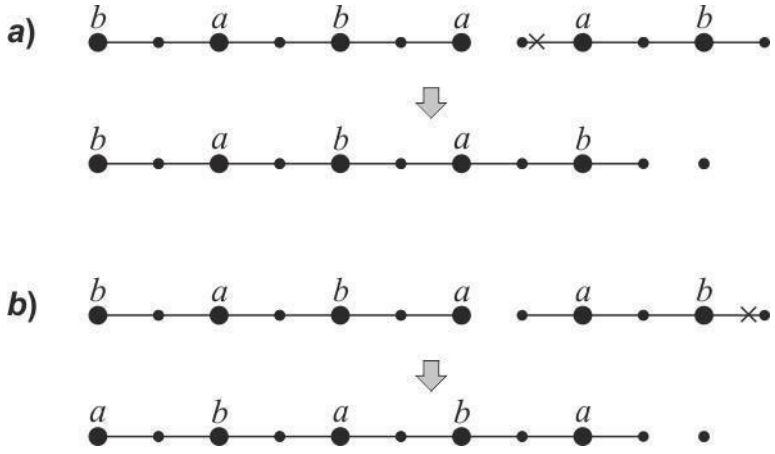
these steps, c is set equal to either a or b for the whole chain of operations preceding this step. This trick ensures that the algorithm has no more than one branching, which emerges when the type 1_c is assigned.



3.2. $2a+3b=1_b$, $2b+3a=1_a$, $2b'+3a=1_a$, $2b+3a'=1_a$, and $2b'+3a'=1'$. Hereafter, the algorithm execution is described for the first case only (other cases are similar): cut an external edge in the $3b$-path and join the special node with the extreme special node of the $2a$-path.



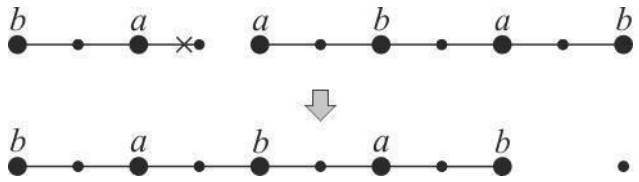
3.3. $2+3=1_c$. Cut an external edge in the 3-path and join the special node with the extreme special node of the 2-path. This results in a path of type 1_a or 1_b, i.e., of type 1_c, depending on which of two external edges was cut.



3.4. $1b+2a+3=2+3=1_c$, $1a+2b+3=2+3=1_c$, and $1a+2b'+3=2+3=1_c$. First carry out the $1b+2a=2$ operation (see below) and then the $2+3=1_c$ one.

3.5. $1a+3b+2=3+2=1_c$, $1b+3a+2=3+2=1_c$, and $1b+3a'+2=3+2=1_c$. First carry out the $1a+3b=3$ operation (see below) and then the $2+3=1_c$ one.

3.6. $1a+2=2a$ and $1b+2=2$b. Cut an external edge in the $1a$-path and join the special node with the extreme special node of the 2-path.



3.7. $1a+3=3a$ and $1b+3=3b$. Cut an external b-edge in the 3-path and join the special node with the extreme special node in the $1a$-path.

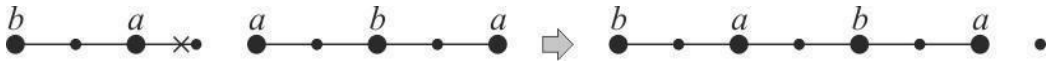
3.8. $1a+1a+2b+3b=2+3=1_c$, $1a+1a+2b'+3b=2+3=1_c$, $1b+1b+2a+3a=2+3=1_c$, and $1b+1b+2a+3a'=2+3=1_c$. First carry out the $1a+2b=2$ and $1a+3b=3$ operations; then the $2+3=1_c$ one.

3.9. $1a+1a+2b=3a+2b=1_a$, $1a+1a+2b'=3a+2b'=1_a$, and $1b+1b+2a=3b+2a=1_b$. First carry out the $1a+1a=3a$ operation (see below); then $2b+3a=1_a$ one.

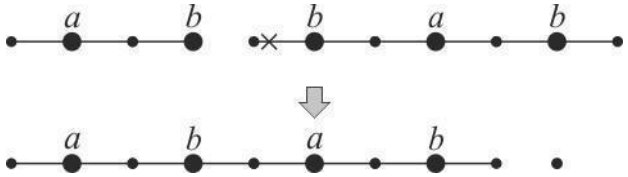
3.10. $1a+1a+3b=1a+3=3a$, $1b+1b+3a=1b+3=3b$, and $1b+1b+3a'=1b+3=3b$. First carry out the $1a+3b=3$ operation; then the $1a+3=3a$ one.

3.11. $1a+1a=3a$ and $1b+1b=3b$. Connect the extreme special nodes of two $1a$-paths.

3.12. $1a+2b=2$, $1a+2b'=2$, and $1b+2a=2$. Cut an external edge in the $1a$-path and join the special with the extreme special node of the $2b$-path.



3.13. $1a+3b=3$, $1b+3a=3$, and $1b+3a'=3$. Cut an external edge in the $3b$-path and join the special node with the extreme special node of the $1a$-path.



3.14. $2a+2b+3+3=2+3=1_c$ and $2a+2b'+3+3=2+3=1_c$. First carry out the $2a+2b+3=2$ operation (see below); then the $2+3=1_c$ one.

3.15. $3a+3b+2+2=3+2=1_c$ and $3a'+3b+2+2=3+2=1_c$. First carry out the $3a+3b+2=3$ operation (see below); then the $2+3=1_c$ one.

3.16. $2a+3+3=1a+3=3a$, $2b+3+3=1b+3=3b$, and $2b'+3+3=1b+3=3b$. First carry out the $2a+3=1a$ operation (see below); then the $1a+3=3a$ one.

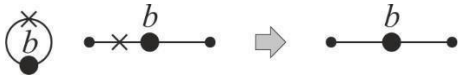
3.17. $3b+2+2=1b+2=2b$, $3a+2+2=1a+2=2a$, and $3a'+2+2=1a+2=2a$. First carry out the $3b+2=1b$ operation (see below); then the $1b+2=2b$ one.

3.18. $2a+2b+3=2a+1b=2$ and $2a+2b'+3=2a+1b=2$. First carry out the $2b+3=1b$ operation; then the $1b+2a=2$ one.

3.19. $3a+3b+2=3a+1b=3$ and $3a'+3b+2=3a'+1b=3$. First carry out the $3b+2=1b$ operation; then the $1b+3a=3$ one.

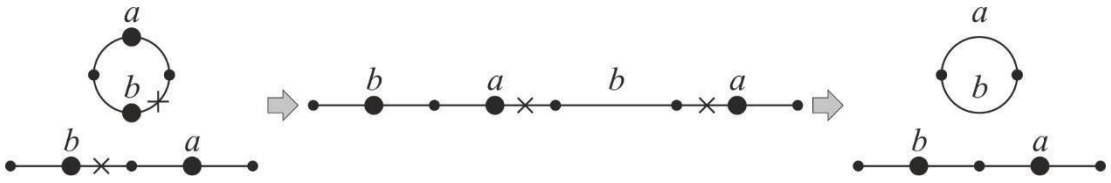
Step 4. If the weight of the double-cut-and-paste operation is greater than the weight of the sesqui-cut-and-paste, the actions 4.1–4.24 are repeated as long as they are applicable; otherwise the steps 4.1′–4.24′ are repeated analogously.

4.1. “Loop”+(“circle” or “path *K* with a *b*-node”) = “circle” or “path of the same type as *K*,” correspondingly. Join the loop node with the *b*-node by double-cut-and-paste (if *K* is not an isolated special node) or by sesqui-cut-and-paste (otherwise).



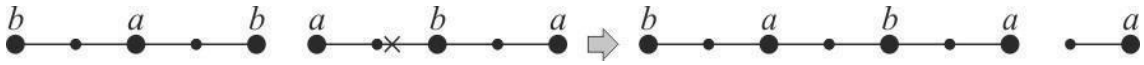
4.1′. Same as 4.1.

4.2. “Circle”+(“circle” or “path *K* with *b*- and *a*-nodes) = “circle” or “path of the same type as *K*.” Insert the circle (by double-cut-and-paste combining two *b*-nodes) near the *b*-node from *K* on the side of the *a*-node; cut out the resulting common edge.



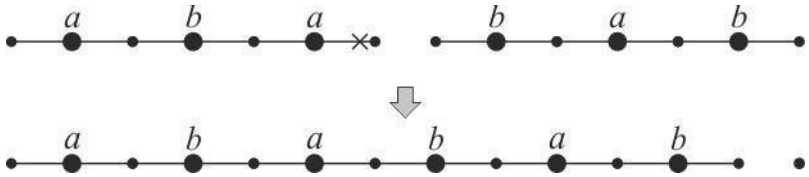
4.2′. Same as 4.2.

4.3. 2*a*+2*b*=2+1′. Cut out two 2*b*-path nodes (the extreme special *a*-node and the neighboring common node) by sesqui-cut-and-paste and join the resulting terminus with the extreme special *b*-node of the 2*a*-path.



4.3′. 2*a*′+2*b*=2+1′.

4.4. 3*a*+3*b*=3. Cut an external edge in the 3*a*-path and join the special node with the extreme common node of the 3*b*-path.

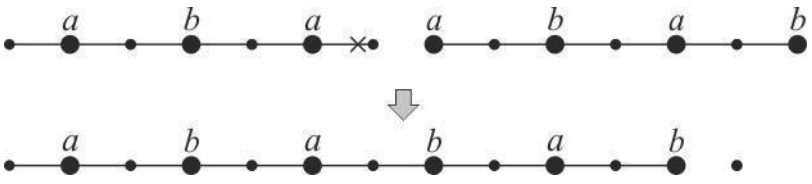


4.4′. 3*a*+3*b*′=3.

4.5. 2*a*+3=1*a* and 2*b*+3=1*b*. Cut an external *b*-edge in the 3-path and join the special node with the extreme special node of the 2*a*-path.

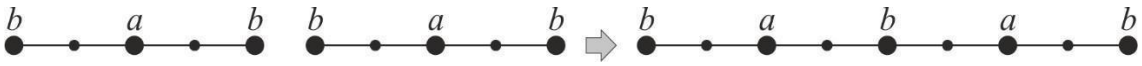
4.5′. 2*a*′+3=1*a*.

4.6. 3*a*+2=1*a* and 3*b*+2=1*b*. Cut an external edge in the 3*a*-path and join the special node with the extreme special node of the 2-path.



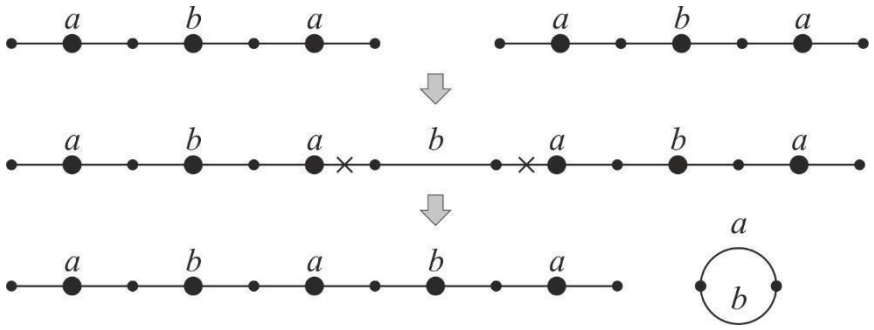
4.6′. 3*a*+2′=1*a*, 3*b*′+2=1*b*.

4.7. 2*a*+2*a*=2*a* and 2*b*+2*b*=2*b*. Join the extreme special nodes of the two paths.



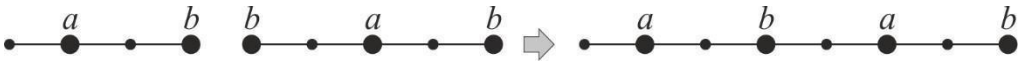
4.7′. $2a'+2a=2a$.

4.8. $3a+3a=3a$ and $3b+3b=3b$. Connect two extreme common nodes of the paths by a common edge, and then cut out this edge.



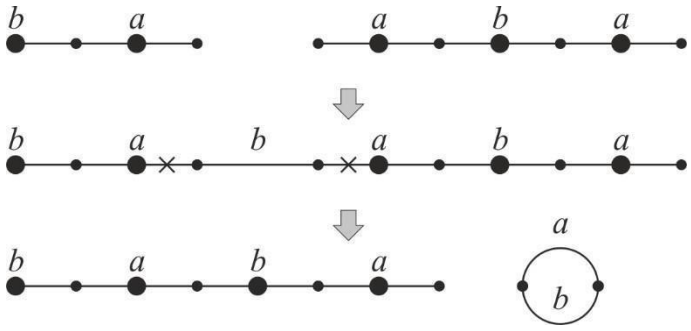
4.8′. $3b'+3b=3b$.

4.9. $1a+2a=1a$ and $1b+2b=1b$. Connect the extreme special nodes of the two paths.



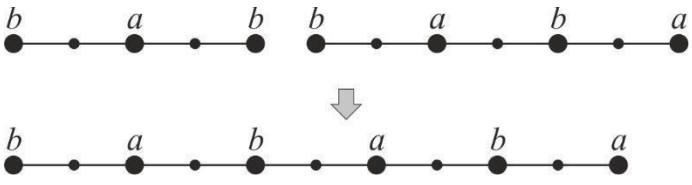
4.9′. $1a+2a'=1a$.

4.10. $1a+3a=1a$ and $1b+3b=1b$. Connect two extreme common nodes of the paths by a common edge, and then cut out this edge.



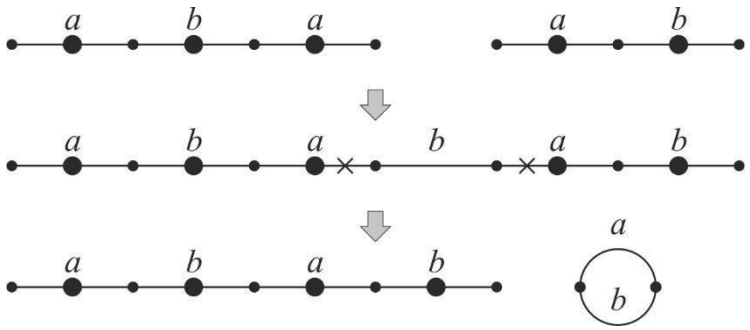
4.10′. $1b+3b'=1b$.

4.11. $2a+2=2$ and $2b+2=2$. Connect the extreme special nodes of the two paths.



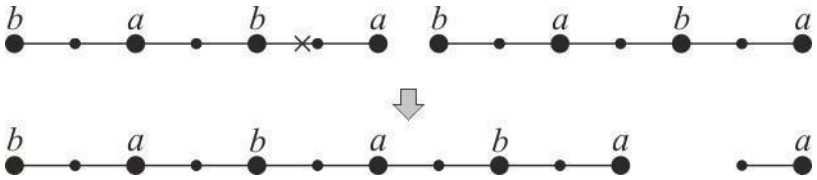
4.11′. $2a'+2=2$, $2a+2'=2$, $2b+2'=2$.

4.12. $3a+3=3$, $3b+3=3$. Connect two extreme common nodes of the paths by a common edge, and then cut out this edge.



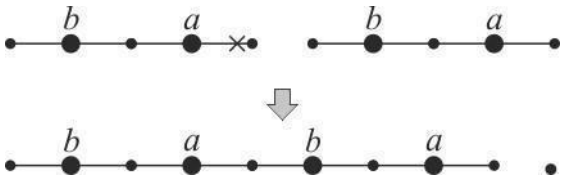
4.12′. $3b'+3=3$.

4.13. 2+2=2+1'. Perform the sesqui-cut-and-paste operation with cutting out the two nodes of the 2-path (the extreme special *a*-node and the neighbor common node) and joining the resulting terminus with the extreme special *b*-node of the other 2-path.



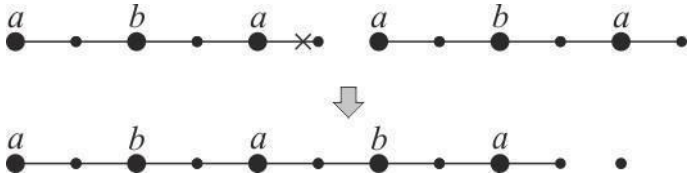
4.13′. $2'+2=2+1'$.

4.14. 3+3=3. Cut an external *a*-edge in the 3-path and join the resulting terminus with the *b*-terminus of the other 3-path.



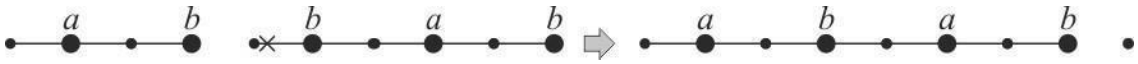
4.14′. Empty action.

4.15. $1_a+1_a=1_a$, $1_b+1_b=1_b$, and $1_b+1_c=1_b$ (set $c=b$). Cut an external edge in the 1_a-path and join the special node with the extreme special node of the other 1_a-path.



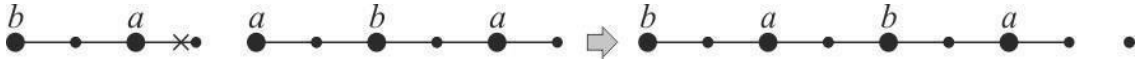
4.15′. $1''+1_b=1_b$, $1''+1_c=1_b$ (set $c=b$).

4.16. $1a+1_b=1a$, $1b+1_a=1b$, and $1a+1_c=1a$ (set $c=b$). Cut an external edge in the 1_b-path and join the special node with the extreme special node of the $1a$-path.



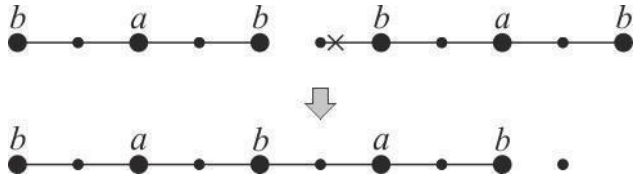
4.16′. $1a+1''=1a$.

4.17. $1a+1_a=1a$, $1b+1_b=1b$, and $1b+1_c=1b$ (set $c=b$). Cut an external edge in the $1a$-path and join the special node with the extreme special node of the 1_a-path.



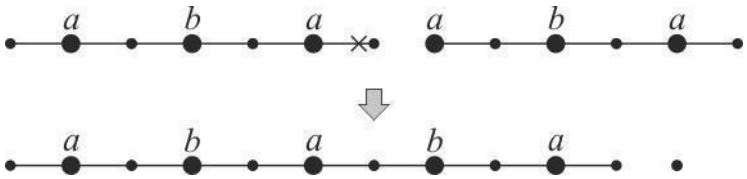
4.17′. $1b+1''=1b$.

4.18. $2a+1_b=2a$, $2b+1_a=2b$, and $2a+1_c=2a$ (set $c=b$). Cut an external edge in the 1_b-path and join the special node with the extreme special node of the $2a$-path.



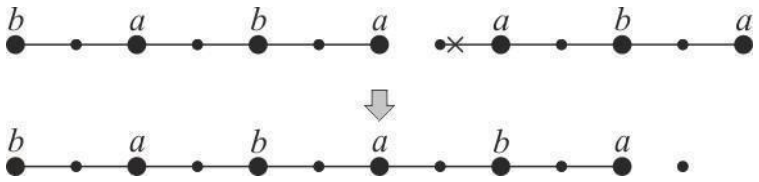
4.18′. $2a'+1_b=2a$, $2a+1''=2a$, $2a'+1_c=2a$ (set $c=b$).

4.19. $3a+1_a=3a$, $3b+1_b=3b$, and $3b+1_c=3b$ (set $c=b$). Cut an external edge in the $3a$-path and join the special node with the extreme special node of the 1_a-path.



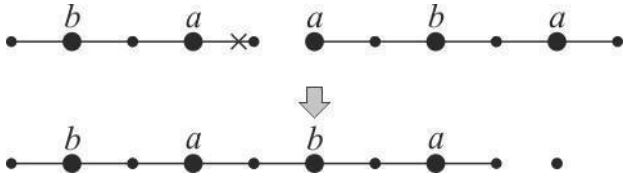
4.19′. $3b'+1_b=3b$, $3b+1''=3b$, $3b'+1_c=3b$ (set $c=b$).

4.20. $2+1_a=2$, $2+1_b=2$, and $2+1_c=2$ (set $c=b$). Cut an external edge in the 1_a-path and join the special node with the extreme special node of the 2-path.



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

4.21. $3+1_a=3$, $3+1_b=3$, and $3+1_c=3$ (set $c=b$). Cut an external edge in the 3-path and join the special node with the extreme special node of the 1_a-path.



4.21′. $3+1''=3$.

4.22. $1_a+1_c=1_a$, $1b+1_c=1b$, $1a+1_c=1a$, $2b+1_c=2b$, and $3a+1_c=3a$. In all cases, set $c=a$.

4.22′. Empty action.

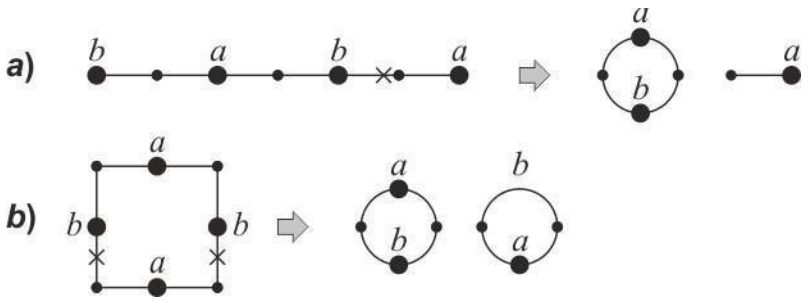
4.23. For the remaining paths of type 1_c, set $c=b$ and perform the $1_b+1_b=1_b$ operation.

4.23′. Empty action.

4.24. Paths, that have hanging edge, are being enclosed into circles by join operation (for paths $2a$, $2b$, $3a$, $3b$), by sesqui-cut-and-paste operations with joining of special nodes (paths 1_a, 1_b, 1_c, 2) or without joining (paths $1a$, $1b$, 3). When enclosing path 1_c we set $c=b$. When enclosing path 2 we choose variant when two b-nodes are joined, after that we delete a-node from 1', see Fig. *a* below. We also cut out the common edge from circles, that were produced by $3a$ or $3b$ closure, then we apply step 4.2 to these circles again.

4.24′. Same as 4.24.

Step 5. Remove isolated special nodes and loops. Cut out 2-circles (using double-cut-and-paste) from circles without common edges to combine two b-nodes (accordingly, the a-node is included into the 2-circle), fig. b. Delete special nodes from the 2-circles.



The end of algorithm description.

Installation and execution of ChromoGGL utilities on Windows

ChromoGGL command line utility is implemented using C++. 32-bit version of executable module is available for download. Utility does not require installation.

Several steps are required to start use utility:

download archive *chromo.zip* and unpack it to any directory (for example, f:/Chromo); run console Windows (Start > Run > cmd) and enter the directory with archive content:

f:

cd f:/Chromo;

run command: chromoggl –h.

In case of successful execution short instruction should appear on the screen. Otherwise the information about the error will be printed. It is recommended to test utility with run_example_*X*.bat where *X* runs from 1 to 5.

Input data for Ghromoggl

The **input data** consists of one file – string with the following data: the *number* of chromosome structures, *name* of specie/culture; the number of chromosome, included to the structure from this specie, that is, the *number* of paths and cycles in the chromosome; *label* (*L*) if chromosome is linear, and (*C*) if cyclic; the *number* of genes in the chromosome; *sequence* of genes in the chromosome, it consists of gene names. If gene is located on the minus-strand, «–» is written in front of its name, otherwise «+»). For example, the file for two structures from Fig. 1 is presented below.

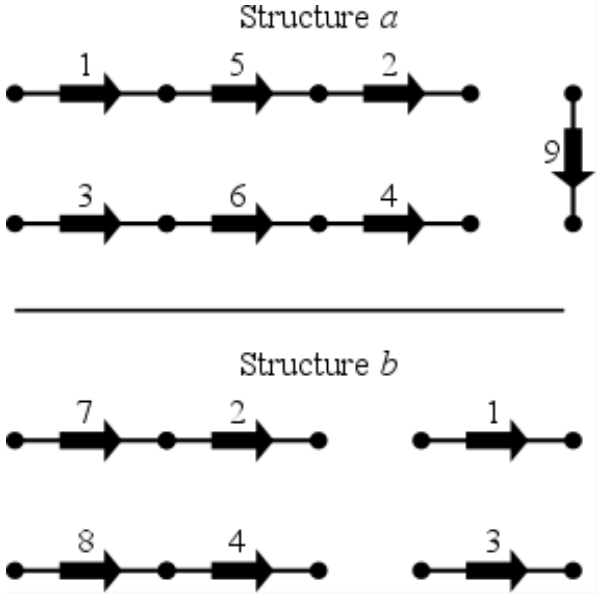


Fig. 1

File with structures, shown on Fig. 1:

2; Structure_*a*; 3; *L*3: +1+5+2; *L*3: +3+6+4; *L*1: +9; Structure_*b*; 4; *L*2: +7+2; *L*2: +8+4; *L*1: +1; *L*1: +3

Joint graph of two given on the Fig. 1 structures, where the special nodes are represented as circles of bigger size, other nodes are circles of smaller size:

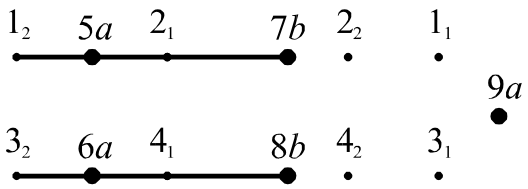


Fig. 2

File with operations costs

Weights are placed one by one in the single string, all of them are represented by one positive double number:

DP = …, SP = …, J = …, C = …, *a*D = …, *b*D = ….

Example of the file – string with operations weights:

DP = 1.2, SP = 1.1, J = 1, C = 0.9, *a*D = 0.8, *b*D = 1.5

File with costs of distances of various types

Weights are indicated for usual and joint distances between two structures.

Example of the file:

```
[weights]
special=1
breakpoint=0.1
```

ChromoGGL parameters

Chromo parameters

The utility requires several parameters for execution, the meaning of these parameters is revealed below. Each parameter has a key (one symbol or several ones) and then an argument. Necessary parameters are -c, -o, -m; for the others parameters there are values by default. A quick reference about parameters is issued when the program is run with the key -h (--help).

Command line parameters

- -c (--chrom) – a name of the file with structures (a path can be included);
- -o (--oper) – a name of the file with operation costs (a path can be included);
- -m (--mode) – execution mode, it equals dist, if task 1 is being solved; tree, if task 2 is being solved;
- -r (--res) – path to the directory with the results of utility execution. The current directory is used by default;
- -d (--coeff) – a name of the file with operation coefficients (a path can be included); file файл distance_weights.ini in the current directory is used by default;
- -f – print matrix in phylip format.

Output data

In task 1 two files are being created: *joint_graph* – the description of the joint graph and *shortest_sequence* – the description of the shortest sequence of transformations. The *first file* has: heading *joint_graph*, heading cycles (…), which includes the number of cycles in the joint graph, then they are listed; heading paths (…) indicates the number of paths, then they are listed. Each cycle is represented as the sequence of its nodes: common node is represented as a gene name, followed by the number 1 if it is the start of gene, and 2 if it is the end. Special node consists of the names of special genes from the block, it is followed by the name of structure, to which this edge belongs. The edge between common nodes is represented as «_», after it the name of structure, to which this edge belongs, is printed. The edge to the special node is represented by

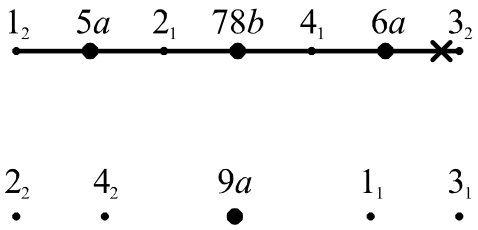
the same symbol, the name of the structure is not printed. At the end of file the number of special a- and b- nodes in the joint graph is printed. It is represented as “spnodes”. If the gene partition is not well-defined, it is enclosed in the brackets.

Example of the first file for the joint graph from Fig. 2:
Joint_graph cycles (0): paths (7):1.1; 2.2; 3.1; 4.2; 9*a*; 1.2_5*a*_ 2.1_7*b*;
3.2_6*a*_4.1_8*b* *a*-spnodes: 3, *b*-spnodes: 2

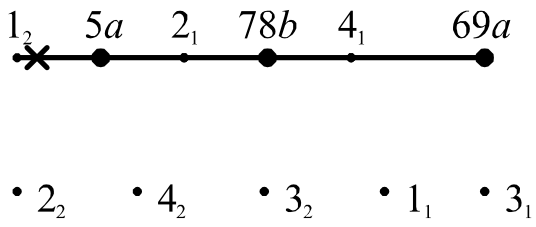
The second file contains the shortest sequence of operations, transforming the joint graph to its final form. Its first line contains: the *number* of operations in the sequence; its total *weight*. Then operations are listed. Each operation contains: component, to which one or several operations are being applied, short names of operations; result.

Example of the second file for the joint graph from Fig. 2:

5; 5.4;
[1.2_5*a*_2.1_7*b* (*L*); 3.2_6*a*_4.1_8*b* (*L*)] C [1.2_5*a*_2.1_78*b*_4.1_6*a*_3.2 (*L*)]
Here special nodes 7*b* and 8*b* from two paths are being united into special node 78*b* at one path. Labels (*L*) and (*C*) mean «path» and «cycle». The result can be seen on the figure below.

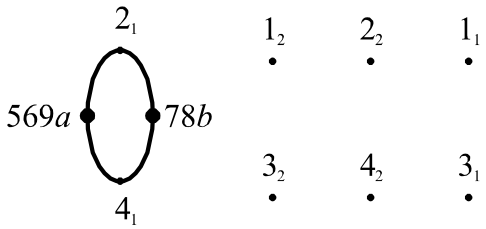


[1.2_5*a*_2.1_78*b*_4.1_6*a*_3.2 (*L*); 9*a* (*L*)] SP [3.2; 1.2_5*a*_2.1_78*b*_4.1_6*a* (*L*)]
Cut of the node 3.2 and union of special nodes 6*a* and 9*a* into one node 69*a*. The result is on the figure below.



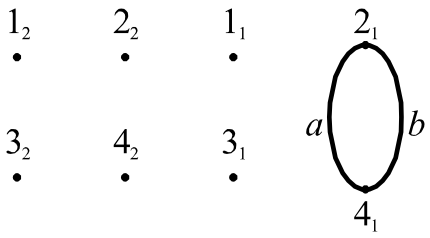
[1.2_5*a*_2.1_78*b*_4.1_6*a* (*L*)] SP [2.1_78*b*_4.1_569*a*_2.1 (*C*); 1.2]

Cut of the node 1.2 and union of the special nodes 5*a* and 69*a* into one node 569*a*. See the result below.



[2.1_78*b*_4.1_569*a*_2.1 (*C*)] *a*D, *b*D [2.1_*a*4.1_*b*2.1 (*C*)]

Deletion of the special nodes 569*a* и 78*b*. See the result below.



Now graph is reduced to the final form, algorithm is over.

Examples of task 1 solutions

The archive *chromo.zip* contains several examples of task 1 solutions. Example – the pair of structures from Fig.1, and the pair of structures from chromosome *Salmonella enterica* to chromosome *Escherichia coli.*

Input data can be found in *artificial* and *Salmonella_Escherichia*, the joint graphs can be found in *artificial* and *Salmonella_Escherichia*, the shortest sequences are described in *artificial* and *Salmonella_Escherichia*. To run the utility on these examples it is necessary to run the following scripts from the archive: *run_example_1.bat* и *run_example_2.bat*.

Also the program was tested using the following 8 sets of artificial data: *artificial1*, *artificial2*, *artificial3*, *artificial4*, *artificial5*, *artificial6*, *artificial7*, *artificial8*. The joint graphs are described in *artificial1*, *artificial2*, *artificial3*, *artificial4*, *artificial5*, *artificial6*, *artificial7*, *artificial8*. The shortest sequences for three variants of weights are described in *artificial1*,

artificial2, *artificial3*, *artificial4*, *artificial5*, *artificial6*, *artificial7*, *artificial8*. The variants are separated from each other by empty line.

The **task 2** solution contains two files: *distance* – matrix of pairwise distances between chromosome structures; *tree* – the evolutionary tree of chromosome structures, that matches best to the matrix. Also if the value of -f parameter is set to 1, two additional files are being written. First one is *infile* – file with matrix in phylip format. The names of species are being replaced with ids from genbank as phylip requires short names. The mapping of names and ids is written to *map.txt* file.

Exampels of task 2 solutions

The first example of the set of chromosome structures contains 66 plastids from rhodophytic branch. All chromosomes are circular.
The second example of the set of chromosome structures contains 18 mitochondrion of class *Aconoidasida*. There are both circular and linear chromosomes in this set.
Input chromosome structures can be found in *rhodophytic_branch* and *mitochondria*, matrixes of pairwise distances are in *rhodophytic_branch* and *mitochondria*, the evolutionary trees are described in *rhodophytic_branch* and *mitochondria*. The archive also contains all this data, suitable for Microsoft Excel.
To run utility on these examples it is necessary to run the following scripts from the archive: *run_example_3.bat* and *run_example_4.bat*.

Recommended standard programs

To work with output data it is recommended to use the following standard programs: Microsoft Excel to work with matrixes. They are printed using tab separated values format (tsv). The dot is used as the separator in numbers. It is necessary to set the correct separator for numbers, it can be done through the menu: File>Options>Additional>Use system separators.
TreeViewX to visualize trees in the Newick format.
Any text editor to work with the joint graphs and the shortest secuences.

Reference

[1] K.Yu. Gorbunov, R.A. Gershgorin, V.A. Lyubetsky. Rearrangement and Inference of Chromosome Structures. *Molecular Biology*, 2015, Vol. 49, No. 3, pp. 327–338