ORTHOCLUSTER USER MANUAL
26/11/2007

1) Introduction

Synteny blocks composed of two or more ortholog genes has been shown to be conserved among species. These blocks are a result of speciation from a common ancestor, and the conservation of their structure and functionality through natural selection reveals that they can be essential in finding key biological processes and devastating diseases, genome rearrangement events and breakpoints, and cis-regulatory elements, among other biological features.

Given the relevance of synteny blocks, a program becomes necessary in order to understand the evolutionary activities that yield speciation between species, and to find these blocks considering different level of conservation between them, this is, handling orientation, strandness and mismatches as the user considers more appropriate for the time of divergence and other characteristics of the species being analyzed.

Orthocluster is a fast and easy-to-use program that provides these functionalities. The program takes as input *n* genomes and the corresponding orthologs, and yields the synteny blocks found among them. When used between two genomes, it reports genome rearrangements such as insertions, transpositions, inversions and reciprocal translocations. Also, Orthocluster can be used to find duplicated blocks within genomes. Finally, Orthocluster can be used for any type of marker for which a one-to-one or one-to-many relation could be establish.

2) Installation and Contact

Orthocluster is available at http://<WEBSITE>. If you have any comments or questions, please email Xinghuo Zeng (xzeng.sfu@gmail.com) and cc Dr. Jack Chen (chenn@sfu.ca), Dr. Jian Pei (jpei@cs.sfu.ca) and Ismael Vergara (iav@sfu.ca).

2.1) Installation in Linux

Extract the downloaded file using the following command in a prompt:

```
% tar xvfz Orthclu.tar.gz
```

The already compiled version of the program found at BIN directory can be used. Also, the source code in the SOURCE directory can be compiled as follows:

```
% make
```

g++ compiler must be installed in order to compile the program successfully.

2.2) Installation in Windows

- Extract the downloaded file.
- Go to the Windows directory. The executable file (OrthoCluster.exe) will be available and ready for usage.

3) Methods

3.1) The problem

Given a set of genomes and a mapping between ortholog genes among these genomes, OrthoCluster is designed to find *synteny blocks*, this is, chromosomal locations in which blocks of genes are preserved, holding the following properties:

a) the total number of ortholog genes in each block is less than ***u***;

b) the number of ortholog genes that have no correspondence in other block(s) of other genome(s) (called inmap mismatches) is less or equal than ***i***;

c) the percentage of ortholog genes within a cluster that have no correspondence in other block(s) of other genome(s) is less or equal than ***ip***;

d) the number of non-ortholog genes (called out-map mismatches) within a block is less or equal than ***o***;

e) the percentage of genes within a block that have no mapping is less than ***op***;

where ***u***, ***i***, ***ip***, ***o*** and ***op*** are user-defined parameters.

The user may also require that the order/strandedness of genes in the block is conserved among all the genomes under analysis. We call such blocks order/strandedness preserved blocks. Please refer to Chapter 4 for more details.

3.2) Algorithm for searching synteny blocks

Orthocluster uses a depth-first search method to identify synteny blocks among genomes. Here we give a brief introduction of the algorithm. For simplicity, we will consider only two genomes, G1 (the reference genome) and G2 (the target genome) and two parameters: the maximal number of ortholog genes included in a block, ***u***, and the maximal number of mismatched ortholog genes in a block, ***i***.

If we could enumerate all subsets of genes shared by the two input genomes G1 and G2, check whether the in-map mismatched genes are less than ***i***, and then output all the maximal sets, then we would find all qualified sets of genes as blocks.

Let {g1, g2, g3, …gn} be the set of ortholog genes of an input genome G1 (or G2). All gene blocks can be divided into the following exclusive groups:

Blocks having g1;
Blocks having g2, but do not have g1;
Blocks having g3, but do not have g1 and g2;
……

The group consisting of blocks having g1 can be further divided into the following groups:

Blocks only having g1;
Blocks having g1 and g2;
Blocks having g1 and g3, but do not have g2;
Blocks having g1 and g4, but do not have g1 and g2;
……

The process is illustrated by a set enumeration tree (Figure 1).

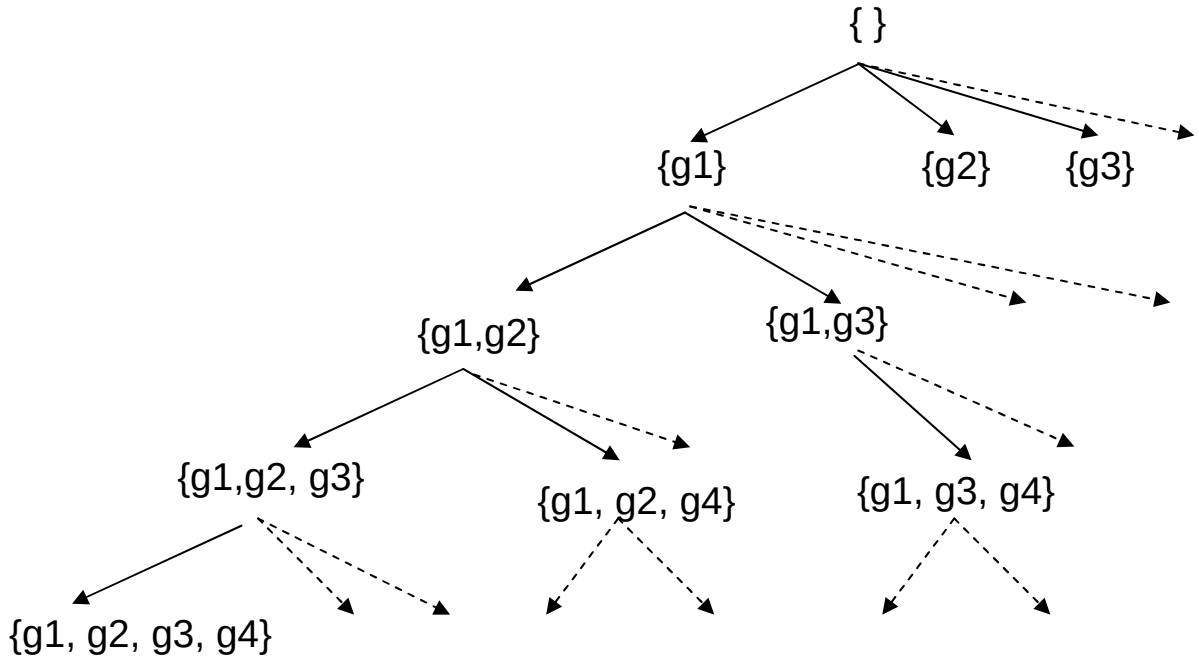


Figure 1. A Set Enumeration Tree. Each node in the tree here represents a subset of (ortholog) genes, each node is obtained by adding one gene to its parent. The descendants of node {g1} in the tree consists of all subsets having g1, the descendants of node {g2} consists of all subsets having g2 but does not have g1, etc.

However, exhaustively enumerating all the subsets requires 2^n steps, where n is the number of ortholog genes in the block. When n is large, the runtime is prohibitive. To reduce the runtime, we apply three pruning techniques, namely sliding window, iterative refining and pruning by in-between genes, which help avoid searching every node in the set enumeration tree.

3.2.1) Sliding window

Suppose the upper bound on the number of ortholog genes in each block defined by the user is **u**. Then, any two in-map genes whose distance is larger than **u** cannot co-exist in one block.

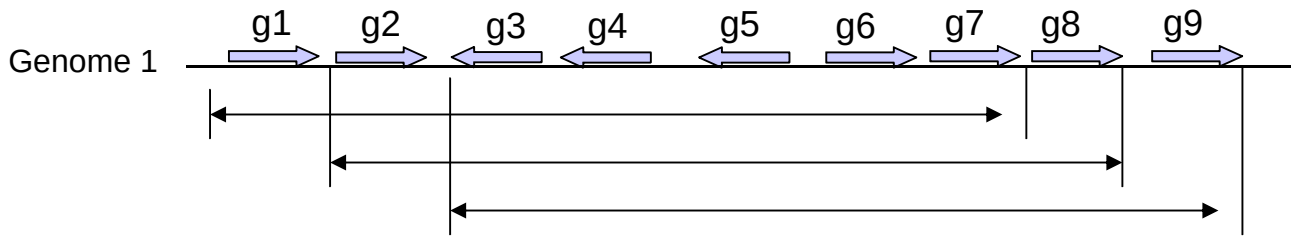


Figure 2. Sliding window of size 7 on Genome 1. Each arrow represents one gene, and the direction of arrows show the orientation (strandedness) of genes. A sliding window of size 7 includes 7 consecutive genes in the genome.

Suppose u=7. Then, g1 and g8 cannot co-exist in one block, and neither can g2 and g9, (Fig. 2). Here, all possible genes that can be combined with g1 to form a block are g2,

g3, g4, g5, g6 and g7. We call {g2, g3, g4, g5, g6, g7} the tail of {g1}, denoted by Tail({g1}). When searching for all blocks having {g1}, we only need to consider combinations of genes consisting of g1 and some genes in tail({g1}); any combination having other genes not included in Tail({g1}) are excluded from our attention.

3.2.2) Iterative refining.

With the sliding window, we can generate the initial tail of a gene. However, if the size of the sliding window is large, the tail could contain several genes, which may still make the search impractical. So we further propose the iterative refining technique to reduce the tail. Suppose the algorithm is currently searching blocks having g1, i.e., searching the sub-tree of g1 in the set enumeration tree, and suppose **u**=7, and ***i***=1.

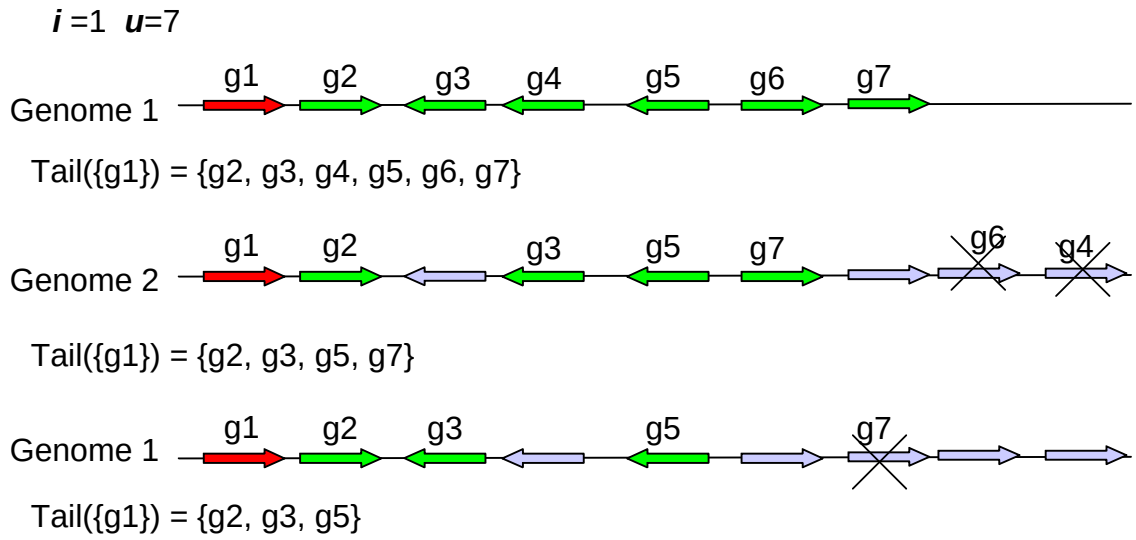


Figure 3. Iterative refining example. The tail of every node in the set enumeration tree can be reduced using the iterative refining technique. We can remove some genes from the tail if adding them to the tail will cause the in-map mismatched genes to exceed **i** in one some genome. We iterative examine all input genomes until no genes can be removed from the tail any more.

From genome 1, we can generate {g2, g3, g4, g5, g6, g7} as the tail of g1. Now let's look at genome 2 and consider the genes beginning from those nearer to g1 to those more distant from g1. g2 is the nearest gene to g1, adding g2 to the block does not result in any mismatched gene in the block. However, adding g3 to the block results 1 mismatched gene, which is equal the maximal number of mismatched genes. Adding g6 to the block will increase the number of mismatched genes to 2, which exceeds **i**, so any genes farther from g7 should not appear in the same block as g1. Thus, we can remove g4 and g6 from the tail of {g1}, resulting in a smaller tail {g2, g3, g5, g7}. The same refining process can also be applied to genome 1 again. Finally only g2, g3 and g5 are retained in the tail of g1. This process is illustrated by Figure 3.

Iterative refining can effectively reduce the size of the tail, and thus reduce the number of nodes in the set enumeration to be searched.

3.2.3) Prune by in-between genes

In the set enumeration tree, when searching the H subtree, let Tail(H) be the set of genes in the tail of H. If there is a gene g such that in any input genome, one the following condition holds:

$$1.\forall g1 \in H \forall g2 \in Tail(H)(pos(G,g1) < pos(G,g) < pos(G,g2) \vee pos(G,g2) < pos(G,g) < pos(G,g1))$$
$$2.\exists g1 \in H \exists g2 \in H(pos(G,g1) < pos(G,g) < pos(g2))$$

where pos(G, g) denotes the position of g in a genome G. Then, if there is no block having H U{g}, there is no block which is a superset of H.

Figure 4 shows an example of pruning by in-between genes.

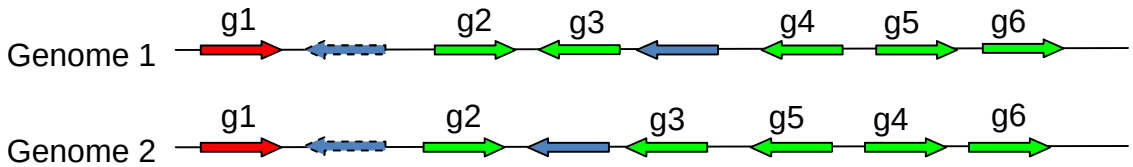


Figure 4. Example of pruning by in-between genes. Suppose Tail ({g1})={g2, g3, g4, g5, g6}. If there is not any cluster having g1 and g2, then there cannot be clusters having g1 and g3 but not g2. This is because if g2 is not considered as a match, the mismatched genes in clusters having g1 and g3 would increase, thus cannot be qualified as a cluster. Similarly, there cannot be any cluster having g1 and g4, but not g2 nor g3, etc.

After applying the above pruning techniques, the search space is significantly reduced. Our experimental results show that in most cases OrthoCluster can perform search efficiently.

3.3) Genome rearrangements

Orthocluster finds the following genome rearrangements when comparing **two** genomes:

3.3.1) Reciprocal translocations

Definition: two non-homologous chromosomes that exchange chunks of DNA by recombination (Fig.5).

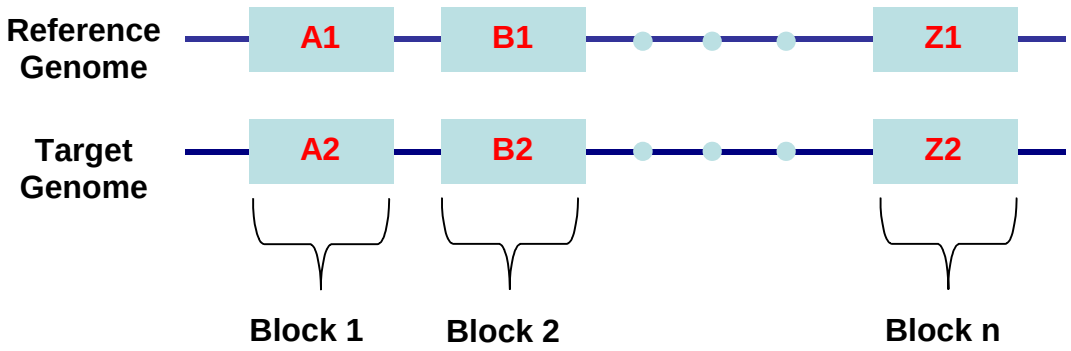


Figure 5. Detection of reciprocal translocations. Each non-nested block found accounts for a reciprocal translocation. The total number of blocks then accounts for the total number of reciprocal translocations,

in this cases *n*. Please note that, depending on the parameters used when running Orthocluster, this number will vary.

3.3.2) Transpositions

Definition: a chunk of DNA excises from one chromosome and inserts into a non-homologous chromosome (Fig. 6).

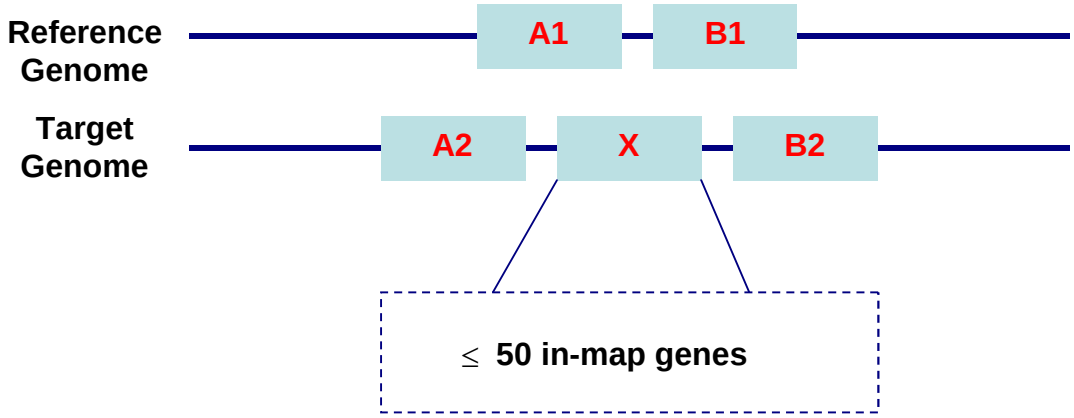


Figure 6. Detection of transpositions. For each adjacent pair of non-nested blocks A_1 and B_1 in the reference genome, a gene block X of size ≤ 50 in-map genes is searched between the corresponding blocks in the target genome, A_2 and B_2. If a cluster is found between A_2 and B_2, then a transposition has been identified.

3.3.3) Inversions

Definition: a DNA segment is inverted in the genome (Fig. 7).

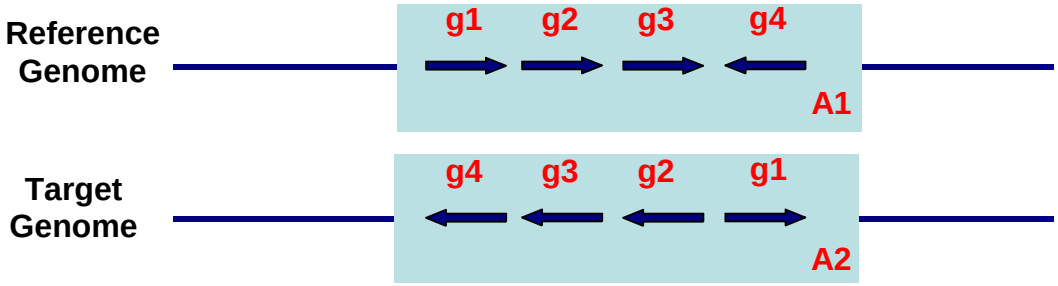


Figure 7. Detection of inversions. For each non-nested block A_1 in the reference genome, if the order of genes in the corresponding block A_2 in the target genome is reversed, then an inversion is identified.

Please note that inversions are only detected by the program when preserving order. See section 4 for more detail about order-preserving blocks.

3.3.4) Insertions/Deletions

Definition: a DNA segment is inserted into a genomic region or deleted from, a genomic region (Fig. 8).

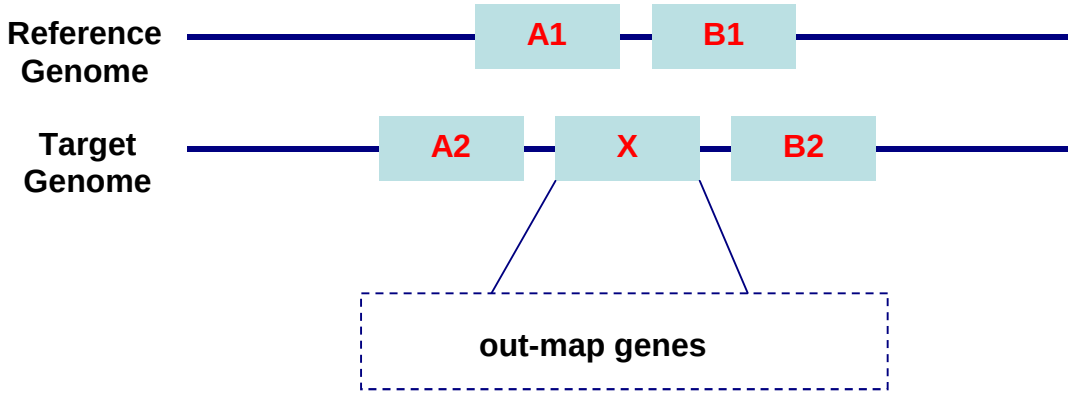


Figure 8. Detection of inversions/deletions. For each adjacent pair of non-nested blocks A_1 and B_1 in the reference genome, out-map genes are searched between the two corresponding blocks in the target genome, A_2 and B_2. If they exist, then an insertion in the target genome has been identified. This can also be considered a deletion in the reference genome.

Note. The user has to consider that the rearrangements are not symmetrical between a pair of genomes, and thus it is necessary to establish a genome as the reference genome, and the other genome as the target genome.

3.4) Syntenic Correlation

A measure of syntenic correlation between two species is computed using the expression

$$\rho = \frac{\sum_{i=1}^{r}\sum_{j=1}^{c}\left(n_{ij} - e_{ij}\right)^2}{n \cdot \min\{r-1, c-1\} e_{ij}}$$

$$0 \le \rho \le 1$$

$\rho = 1$ if and only if, for one of the two species, knowing which chromosome an ortholog belongs to in that species determines which chromosome the ortholog is on in the other species.

$\rho = 0$ if and only if the counts of orthologs are perfectly independently scattered on the chromosomes of the two species.

To compute this value, an *r x c* contingency table is created and displayed, summarizing the number of ortholog genes on each of the chromosomes 1 through *r* of the first genome that are on each of the chromosomes 1 through *c* of the second genome.

n_{ij} is the observed number of genes on species *A* chromosome *i* with an ortholog on species *B* chromosome *j*.

e_{ij} is the expected number of genes in each cell assuming that the genes are scattered independently in the two genomes.

4) Parameters

To run Orthocluster in Linux, the following command should be used:

```
% ./Orthclu –m <mapping_file> -g1 <genome_1_file> ... –gn
<genome_n_file> -d <job_name> [OPTIONS]
```

To run Orthocluster in Windows, the following command should be used:

```
% Geneclu.exe –m <mapping_file> -g1 <genome_1_file> ... –gn
<genome_n_file> -d <job_name> [OPTIONS]
```

The mapping file refers to the file containing the ortholog genes. Genes that belong to this file are called in-map genes. Genes that do not belong to this file are called out-map genes. See Table 1 and Table 2 for details about [OPTIONS] .

Table 1. Description of parameters.

Parameter	Description
-u	The upper bound on the number of ortholog genes in each block
-l	The lower bound on the number of ortholog genes in each block
-m	Mapping file name
-g*i*	Sequence file corresponding to the *ith* column in the mapping file
-i	Maximal number of in-map mismatch genes
-o	Maximal number of out-map mismatch genes
-ip	Maximal percentage of in-map mismatch genes
-op	Maximal percentage of out-map mismatch genes
-d	Job id
-gff	Generates .gff file for display in synbrowse
-r	Finds order-preserving blocks
-s	Finds strandedness-preserving blocks only
-rs (-sr)	Finds strandedness and order preserving blocks only
-f	Finds non-overlapping blocks only
-x	Sorts the blocks according to its size
-v	Prints warning information in the STDOUT
-h	Prints help information

Table 2. Values for input parameters.

Parameter	Default Value	Range	Optional?
-u	1000	[1-∞)	Y
-l	1	[1-u]	Y
-m	N.A.	N.A.	N
-gi	N.A.	N.A.	N
-i	0	[0-∞)	Y
-o	0	[0-∞)	Y
-ip	0	[0-100)	Y
-op	0	[0-100)	Y
-d	N.A.	N.A.	N
-gff	OFF	{ON,OFF}	Y
-r	OFF	{ON,OFF}	Y
-s	OFF	{ON,OFF}	Y
-rs (-sr)	OFF	{ON,OFF}	Y
-f	OFF	{ON,OFF}	Y
-x	OFF	{ON,OFF}	Y
-v	OFF	{ON,OFF}	Y
-h	OFF	{ON,OFF}	Y

4.1) About the –i, -ip, -o and –op parameters.

The –ip parameter works as follows. If this parameter is set to a certain value *ip*, with

$$0 \leq ip < 100$$

Then the constraint is exceeded if

In-map-mismatches / in-map-genes in the block > *ip*

Also, a dependency between the –i and –ip parameters exist. A block must satisfy at least one of the –i and –ip constraints. A block violating both constraints is not considered as valid.

The –op parameter works as follows. If this parameter is set to a certain value *op*, with

$$0 \leq op < 100$$

Then the constraint is exceeded if

out-map-mismatches / total number of genes in the block > *op*

The dependency between the –o and –op parameters is the same as the one existing for –i and –ip.

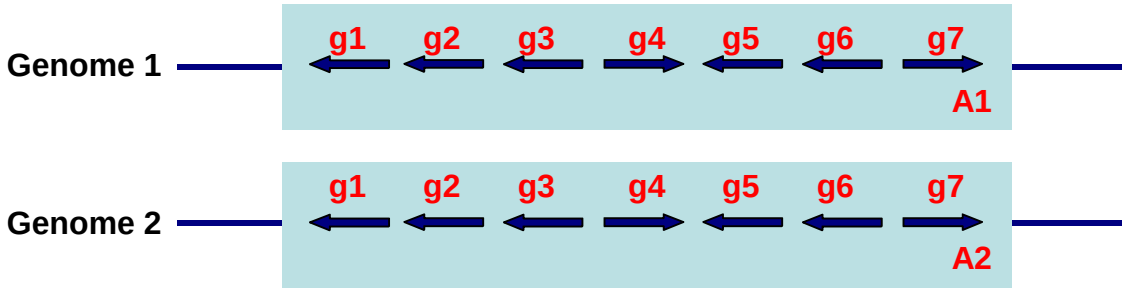
Example 1: Understanding the –i and –o parameters.

Lets assume that a comparison between two genomes is going to be applied. If the user sets

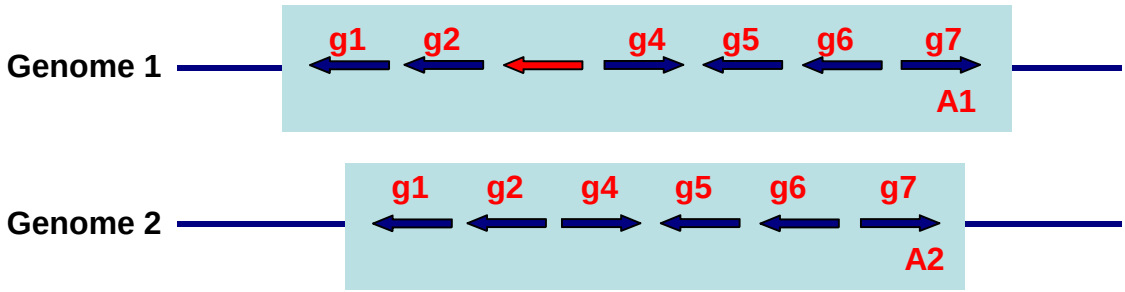
- the upper bound of ortholog genes in each block to –u = 7,
- the lower bound to –l = 1,
- the maximal number of mismatched genes from the mapping file to –i = 1, and
- maximal number of mismatched genes not from the mapping file but existing in the genome file to –o = 1.

Then, for 7 ortholog genes g1, g2, ..., g7, the following blocks are correct:

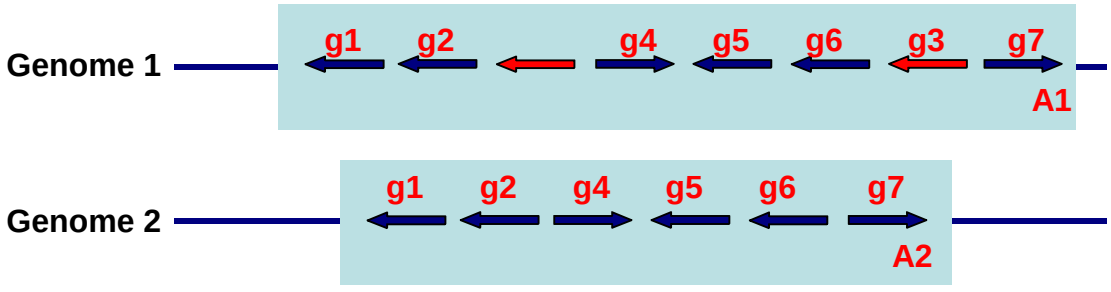
(i) Blocks conformed of the 7 ortholog genes with no in-map mismatches and no out-map mismatches.



(ii) Blocks conformed of 6 ortholog genes. For block A1, one out-map mismatch (in red) occurs.

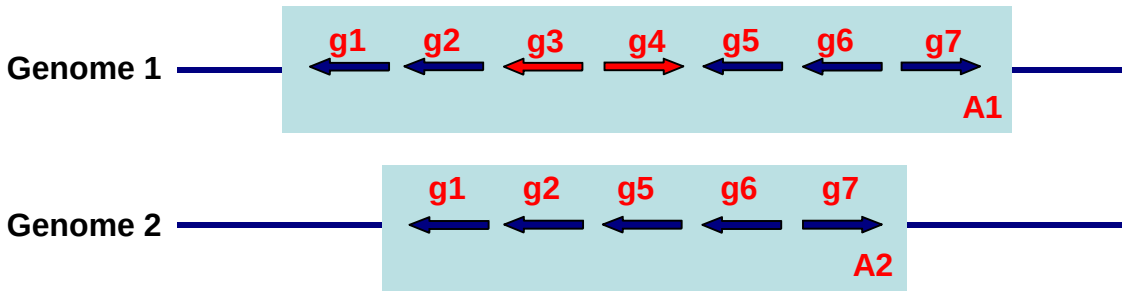


(iii) Blocks conformed of 6 ortholog genes. For block A1, one in-map mismatch (g3) and one out-map mismatch exists.

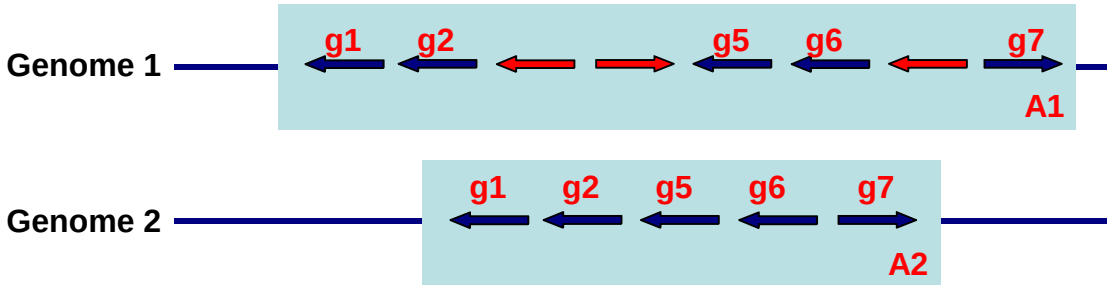


The following cases are not correct:

(iv) The parameter –i is exceeded. There are two in-map mismatches, and a maximum of one is allowed.



(v) The parameter –i is exceeded. There are three out-map mismatches and a maximum of one is allowed.



4.2) About the –r, -s and –rs parameters.

When the parameter –r is used, the program will search for all the synteny blocks that have consistent order, i.e., the order of the genes in each block is the same, or inverted order, i.e., the order of the genes in one block is inverted with respect to the order of the genes in the second block.

When the –s parameter is used, the program will search for all the synteny blocks that have consistent strandedness, i.e., for each pair of orthologs in the blocks the orientation is the same, or reversed strandedness, i.e., for each pair of orthologs in the blocks the orientation is reversed for one gene with respect to the other.

The parameters –r, -s, and –rs (or -sr) allows the user to handle four different types of preserved order and strandedness. Lets consider two genomes, and two blocks in each genome.

a) Consistent order, consistent strandedness

In this case, the order of the genes as well as their strandedness in the blocks of each genome is the same for each pair of orthologs (Fig. 9).

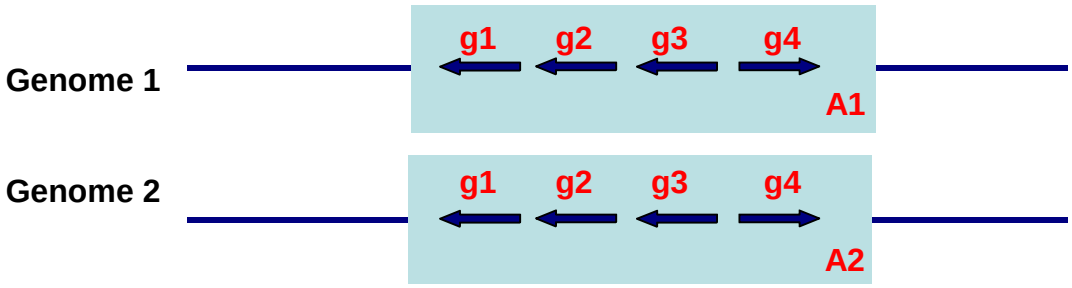


Figure 9. Consistent order and consistent strandedness. In this example, blocks A1 in genome 1 and A2 in genome 2 are composed of four genes. The order of the genes in each block is the same, and each pair of genes has the same orientation. This type of blocks will be found if setting –r and –s together, or –rs.

b) Consistent order, reversed strandedness

In this case, the order of the genes is the same, but strandedness of the genes is reversed for each pair of orthologs (Fig. 10).

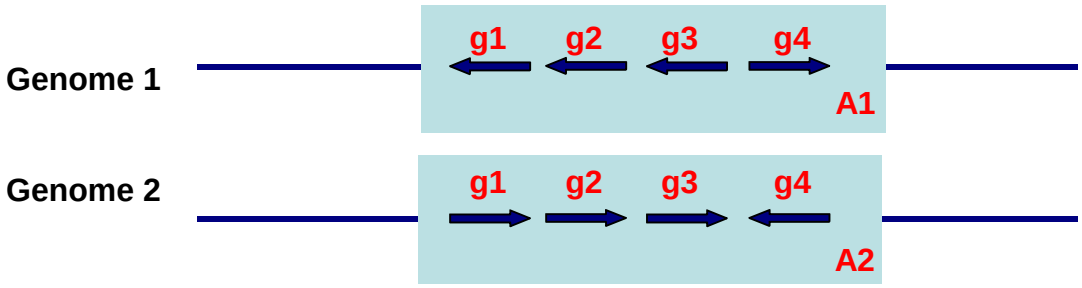


Figure 10. Consistent order, reversed strandedness. In this example, blocks A1 in genome 1 and A2 in genome 2 are composed of four genes. The order of the genes in each block is the same, but each pair of genes has different orientation. This type of blocks will be found if setting –r and –s together.

c) Inverted order, consistent strandedness

The order of the genes is inverted in one block with respect to the other, but their strandedness remains the same (Fig. 11).

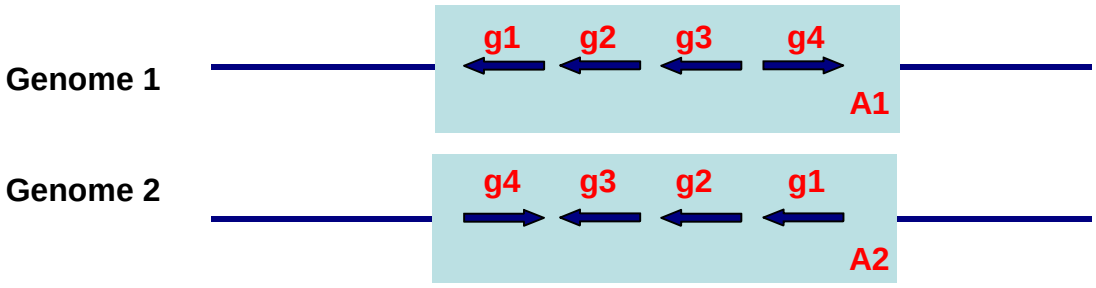


Figure 11. Inverted order, consistent strandedness. In this example, blocks A1 in genome 1 and A2 in genome 2 are composed of four genes. The order of the genes in block A1 is inverted with respect to that in block A2, and each pair of genes has the same orientation. This type of blocks will be found if setting –r and –s together.

d) Inverted order, reversed strandedness.

The order of the genes is inverted in one block with respect to the other, and the strandedness of the genes is reversed for each pair of orthologs (Fig. 12).

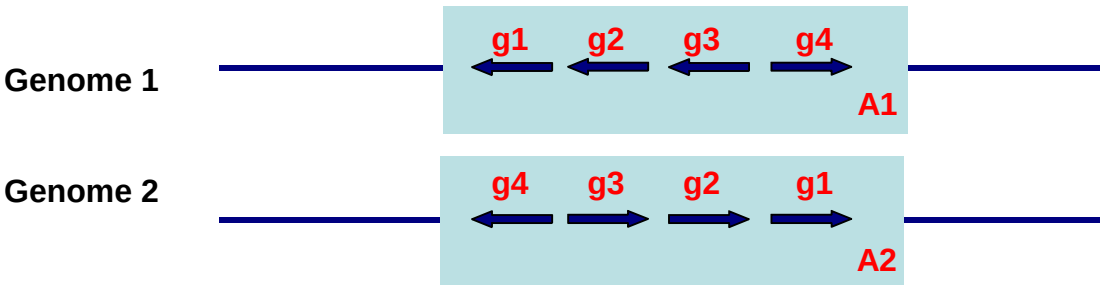


Figure 12. Inverted order, reversed strandedness. In this example, blocks A1 in genome 1 and A2 in genome 2 are composed of four genes. The order of the genes in block A1 is inverted with respect to that in block A2, and each pair of genes has different orientation. This type of blocks will be found if setting –r and –s together, or –rs.

If the user sets the parameters –r –s when running orthocluster, then cases a) through d) will be considered by the algorithm when searching for synteny blocks.

If the user sets -rs then only cases a) and d) will be considered by the algorithm when searching for synteny blocks. Hence, using –rs is a more stringent criteria for searching blocks than using –r –s separately.

Example 2: Running Orthocluster for comparing two genomes.

If 2 genomes will be compared in order to find all possible non-overlapping blocks that preserve strandedness of size 2 and less or equal than 400, then the command is:

```
% ./Orthclu -m <orthologs_file> -g1 <genome_1_file> -g2
<genome_2_file> -d <job_name> -f –s –l 2 –u 400
```

Example 3: Running Orthocluster to find duplications within a genome.

If a genome will be searched for all duplicated regions that preserve order and with size greater or equal than 2, then the command is:

```
% ./Orthclu –m <duplications_file> -g1 <genome_file> -d <job_name> -l
2 –r
```

5) File formats

5.1) Input Files

Orthocluster receives two types of input files:

5.1.1) Genome File

A genome file should be given as input for each species analyzed. The structure of each line composing a genome file is:

```
GENE_NAME   CONTIG      START END   STRAND
```

An example of a genome file is as follows:

```
Y74C9A.3    I     4221  10148 -1
Y74C9A.2    I     11641 16585 1
Y74C9A.4a   I     17911 26778 -1
Y74C9A.5    I     28280 32482 -1
Y74C9A.1    I     43733 44677 1
Y48G1C.12   I     47472 49416 1
Y48G1C.4    I     49921 54360 1
Y48G1C.5    I     55337 64021 -1
C01B12.4    II    6664  9233  1
C01B12.5    II    9808  11826 -1
C01B12.3    II    12986 15858 1
C01B12.2    II    19538 21842 1
C01B12.1    II    23347 24347 1
C01B12.8    II    24771 25381 1
C01B12.7    II    25517 26961 -1
F23F1.4     II    27611 28158 -1
F23F1.5     II    30116 31574 -1
F23F1.3     II    31825 33559 1
F23F1.2     II    34488 35150 1
F23F1.6     II    35301 37302 -1
F23F1.7     II    38605 39487 -1
.
.
```

GENE_NAME has to be unique. Also, isoforms should be removed before using the program. It is not necessary to sort the genes within or between contigs.

5.1.2) Mapping File

The mapping file refers to the established relation known among the different genes in different species (in the case of orthologs), or different genes in the same species (in the case of duplicated genes). Either for orthologs or duplicated genes, the structure for each line composing the mapping file is:

```
GENE_1      GENE_2 … GENE_N
```

An example of a mapping file is as follows:

```
ZK617.1a    CBG06205
```

```
ZK617.1a    CBG06206
C09D1.1a    CBG12070
C41A3.1     CBG14683
K11C4.5     CBG19042
F29D11.1    CBG11882
F33D4.2a    CBG05938
.
.
.
```

Note that in the example ZK617.1a appears two times. This is perfectly allowed by Orthocluster, i.e., it handles one-to-one relations as well as one-to-many relations.

5.2) Output Files

Orthocluster yields as output three files:

(i) a **.cluster** file, that shows the clusters found over genomes,
(ii) a **.rgmt** file, that shows the different types of rearrangements found between two genomes,
(iii) a **.stat** file, that summarizes information related to the size distribution of the clusters and the frequencies of each type of rearrangement, and
(iv) a **.log** file, which collects all the information used and generated while running Orthocluster.

The name of each file is given by the Job ID.

Each file has a common header which has the following structure:

```
Time and date of the current running.

Settings & Parameters:
No. of sequence                     : number of –gi parameters
Max group size                      : -u
Min group size                      : -l
Max out-map-mismatches              : -o
Max out-map-mismatch percentage     : -op
Max in-map-mismatches               : -i
Max in-map-mismatch percentage      : -ip
Find order preserving clusters      : -r
Find strand preserving clusters     : -s
Find non-overlapping clusters       : -f
Mapping file name                   : -m
Sequence 1 file name                : -g1
Sequence 2 file name                : -g2
.
.
.
Sequence N file name                : -gN
Job ID                              : -d
```

An example of this header for a running between two genomes, for finding non-overlapping blocks with a minimum size of 1, a maximum size of 1000, with no mismatches, and preserving the four different types of strandedness and orientation is as follows:

```
Sat Jul 14 17:16:44 2007
```

```
Settings & Parameters:
No. of sequence                                          : 2
Max group size                                           : 1000
Min group size                                           : 1
Max out-map-mismatches                                   : 0
Max out-map-mismatch percentage                          : 0%
Max in-map-mismatches                                    : 0
Max in-map-mismatch percentage                           : 0%
Find order preserving clusters (-r)                      : Yes
Find strand preserving clusters (-s)                     : Yes
Find order and strandedness preserving clusters (-rs): No
Find non-overlapping clusters                            : Yes
Mapping file name                                        : G1_G2_orthologs.txt
Sequence 1 file name                                     : G1_genes.list
Sequence 2 file name                                     : G2_genes.list
Output file name                                         : output
```

5.2.1) The .cluster file

Given N genome, the first line of each block has the following format:

```
Column  1       : cluster ID.
Column  2       : 'N' if it is a nested cluster; empty otherwise.
Column  1+2     : cluster size for genome 1.
Column  2+2     : cluster size for genome 2.
.
.
.
Column  N+2     : cluster size for genome N.
Column  1+(N+2) : in-map mismatch for genome 1.
Column  2+(N+2) : in-map mismatch for genome 2.
.
.
.
Column  N+(2+N) : in-map mismatch for genome N.
Column  1+(2+2N): out-map mismatch for genome 1.
Column  2+(2+2N): out-map mismatch for genome 2.
.
.
.
Column  N+(2+2N): out-map mismatch for genome N.
```

Then, each gene in each genome conforming a block is described by the following format line:

```
Column  1  : gene ID in the block.
             If  it  is  an  in-map-gene  mismatch,  then  no  ID  is
assigned.
Column  2  : position of gene in chromosome.
Column  3  : strand.
Column  4  : chromosome.
Column  5  :

            '*' if it is an in-map mismatch.
            '++' if the gene has multiple matches in the other genome.
            '+' if the gene is one of multiple matches of another gene
            in the other genome.
```

```
            ‘ ’ otherwise.

Column  6  : gene name.
```

If there are more than two genomes being analyzed, then column 5 will suppress ‘++’ and ‘+’ functionality.

An example of a block is shown next:

```
CL-2          13     25           2      8            0      5

  1:      158   -1            I             Y48G8AL.9             11:     1782    1          chrI              CBG08484
  2:      159   -1            I            Y48G8AL.10             11:     1783    1          chrI              CBG08485
  3:      160   -1            I            Y48G8AL.14             10:     1784   -1          chrI       +      CBG08486
  4:      161    1            I            Y48G8AL.8a             10:     1785   -1          chrI       +      CBG08487
  5:      162    1            I             Y48G8AL.7              9:     1786   -1          chrI              CBG08488
  6:      163   -1            I            Y48G8AL.11             1787    1          chrI       *      CBG08489
  7:      164    1            I             Y48G8AL.6             1788    1          chrI       *      CBG08490
  8:      165    1            I             Y48G8AL.5             1789   -1          chrI              CBG08491
  9:      166    1            I             Y48G8AL.1             1790    1          chrI       *      CBG08492
          167   -1            I      *      Y48G8AL.12             8:     1791   -1          chrI              CBG08493
          168   -1            I      *      Y48G8AL.13             7:     1792   -1          chrI              CBG08494
 10:      169    1            I     ++      Y48G8AL.15             6:     1793    1          chrI              CBG08495
 10:      169    1            I     ++      Y48G8AL.15             1794    1          chrI       *      CBG08497
 11:      170   -1            I             Y48G8AR.1              1795   -1          chrI       *      CBG08499
                                                                   1796    1          chrI       *      CBG08500
                                                                   1797   -1          chrI              CBG08501
                                                                   1798    1          chrI              CBG08504
                                                                   1799    1          chrI       *      CBG08506
                                                                   1800    1          chrI              CBG08507
                                                                   1801    1          chrI       *      CBG08508
                                                                    5:     1802   -1          chrI              CBG08509
                                                                    4:     1803   -1          chrI              CBG08510
                                                                    3:     1804    1          chrI              CBG08511
                                                                    2:     1805    1          chrI              CBG08512
                                                                    1:     1806    1          chrI              CBG08513
```

Following the format described above for the header, it is easy to see that:

```
Column  1     : CL-2
Column  2     : ‘ ’
Column  3     : 13
Column  4     : 25
Column  5     : 2
Column  6     : 8
Column  7     : 0
Column  8     : 5
```

5.2.2) The .rgmt file

Each rearrangement is reported as a line with the following format.

```
INSERTION:
Column 1       :       Insertion number.
Column 2       :       Chromosome of clusters in genome 1.
Column 3       :       Cluster IDs.
Column 4       :       Gene range of clusters in genome 1.
Column 5       :       Gene range of inserted genes in genome 1.
```

Example:

```
IS-7    I       CL-18/CL-19     46-46...49-49   47-48
```

```
TRANSPOSITIONS:
Column 1        :       Transposition serial number.
Column 2        :       Cluster IDs.
Column 3        :       Gene range of clusters in genome 1.
Column 4        :       Gene range of clusters  in genome 2.
Column 5        :       Breakpoint.
```

Example:

```
TP-5 CL-29/CL-30  I chrI  67-67...68-70 242-242...190-192  193-241
```

```
INVERSIONS:
Column  1      :       Inversion serial number.
Column  2      :       Cluster ID.
Column  3      :       Chromosome of cluster in genome 1.
Column  4      :       Chromosome of cluster in genome 2.
Column  5      :       Gene range of cluster in genome 1.
Column  6      :       Gene range of cluster in genome 2.
```

Example:

```
IV-7    CL-34   I       chrI    74-78   12461-12457
```

```
RECIPROCAL TRANSLOCATIONS:
Column  1      :       Reciprocal translocation serial number.
Column  2      :       Chromosome of cluster in genome 1.
Column  3      :       Chromosome of cluster in genome 2.
Column  4      :       Gene range of cluster in genome 1.
Column  5      :       Gene range of cluster in genome 2.
```

Example:

```
RT-21   I       chrI    52-55   195-198
```

5.2.3) The .stat file

This file reports two types of statistics:

5.2.3.1) Block Statistics

In this file, the total number of clusters, number of non-nested clusters and number of nested clusters is first reported.

Then, for each genome, the following values are displayed:

```
Smallest cluster size
Largest cluster size
```

```
Median of nested cluster size
Median of non-nested cluster size
```

Also, the size distribution for the nested and non-nested cluster is reported as follows:

```
SIZE    NUMBER_OF_CLUSTERS
1       <number of blocks of size 1>
2       <number of blocks of size 2>
.
.
.
K       <number of blocks of size K>
```

5.2.3.2) Genome rearrangements statistics.

In this section, the following values are first displayed:

```
Number of insertions/deletions
Number of inversions
Number of reciprocal translocations
Number of transpositions
```

Finally, the syntenic correlation value between two genomes and the corresponding contingency table is reported.