lossgainRSL v6.31 User's Manual

Contents

1 General information

The lossgainRSL program is aimed at the prediction of gene losses and gains between several groups of species. It allows the user, for a given *reference* species, to identify its genes that are present in and/or absent from several sets of species in accordance with a given logical function (predicate). The elementary condition "gene *X* is present in the set of species *S*", and the predicate is a Boolean function composed of arbitrary number of such conditions using logical connectives AND, OR, NOT and the parentheses. A set of species does not need to be a taxonomic group; it can be composed on the basis of arbitrary traits. In particular, a species may belong to multiple sets and *S* may consist of a single species. For a gene *X* to present in the set *S*, it is required that *X* is present in at least *p* species from *S* (*p* is a parameter of the condition). The presence of gene *X* of the reference species in some other species is understood as not only presence of gene *X′* that is an ortholog or close homolog of *X* in the genome of other species, but also the *synteny*: there must be several distinct genes ("witnesses") in the vicinity of *X*, which have respective homologous witnesses in the vicinity of *X′*. The sets of species, predicate, required number of witnesses and other synteny details, vicinity sizes and the extent of homology – all are the program parameters, which allows its application to the wide range of studies.

The current version of lossgainRSL is primarily orientated to species and genomic data included in the Ensembl database (http://www.ensembl.org/), but this user's manual also describes other possibilities such as using GenBank (https://www.ncbi.nlm.nih.gov/genbank/), RefSeq (https://www.ncbi.nlm.nih.gov/refseq/), and OrthoDB (https://www.orthodb.org/) data. The program has been written in C++ as a command line utility for Windows/linux. Both 32- and 64-bit OS versions are supported; the latter is highly recommended for studies with many species because of memory requirements. The program supports parallel multiprocessing in the MPI environment of version 2.1 or above.

The lossgainRSL program was developed in the Laboratory of Mathematic Methods and Models in Bioinformatics, Institute for Information Transmission Problems (Kharkevich Institute) of the Russian Academy of Sciences (http://lab6.iitp.ru) by Lev Rubanov, Alexandr Seliverstov and Vassily Lyubetsky. The current versions of lossgainRSL binaries for Windows and the source code for linux (subject to the GNU GPL license) are available at http://lab6.iitp.ru/en/lossgainrsl/ together with the test example and this manual.

Minimum program requirements: Windows or linux OS 32/64 bit (64-bit is recommended), single CPU with the frequency of 1 GHz (3-4 GHz is recommended), 2 GB RAM (8 GB is recommended for multi-species studies).

2 How to install and test

2.1 Windows version

No installer is provided for Windows; the user should download zipped archive(s) with chosen executable and unpack it into a working directory. There are three variants of the executable:

- `lossgainrsl.exe` – intended for 32-bit Windows;
- `lossgainrsl64nompi.exe` – single-CPU version for 64-bit Windows;
- `lossgainrsl64.exe` – parallel version for 64-bit Windows with MS-MPI installed.

The third option requires the Microsoft MPI v10.0 redistributable package (https://www.microsoft.com/en-us/download/details.aspx?id=57467) to be installed in Windows system. Different MPI brands or versions are not supported.

In addition, for the binary executables to operate under Windows, Microsoft Visual C++ for Visual Studio 2013 redistributable package(s) may be required, which are available from: http://download.microsoft.com/download/c/c/2/cc2df5f8-4454-44b4-802d-5ea68d086676/vcredist_x86.exe and http://download.microsoft.com/download/c/c/2/cc2df5f8-4454-44b4-802d-5ea68d086676/vcredist_x64.exe.

The operability testing in Windows is accomplished by the test example with six species that is available at http://lab6.iitp.ru/en/lossgainrsl/example.zip, as follows:

1. Download the archive, `example.zip`, and unpack it to the working directory, where the `example` folder will be created.
2. Copy chosen executable(s) to the `example` folder.
3. Start the Command Prompt and switch to the folder.
4. Depending on the selected variant of the program, run one of the commands:

`run`	for checking the 32-bit version,
`run64`	for checking the 64-bit version without MPI,
`run64mpi`	for checking the 64-bit version with MPI (on two CPUs).

5. After few minutes, the two first options will display the message:
   ```
   Test completed OK.
   ```
   It confirms that respective executable is operable.
6. If the MPI-enabled variant is being checked, the message will read:
   ```
   Xs: Completed OK.
   ```
   The user should compare a contents of the created file, `result.txt`, with that of `result4.ref` from the same folder. It can be done easier by import to Excel. The created file has to contains the same four-line groups, possibly in different order. Discrepancies in the very last column should be ignored.

2.2 linux version

The lossgainRSL program is installed through the building from the source code available for download at the program webpage. In order to use MPI parallelization, an MPI 2.1 compliant version of libraries must be installed in the system. The current version of Open MPI package (https://www.open-mpi.org/) is recommended for various linux versions. Though, many high-performance clusters use MVAPICH or other implementations of the MPI which also will do. To install and test the program, the following steps to be performed:

1. Download the program distribution tarball, usually named `lossgainrsl-`*`x.yy`*`.tar.gz`, where *`x.yy`* is a version number, to a working directory.
2. Unpack the file with a command:
   ```
   tar -xf filename
   ```
3. Switch to the directory created:
   ```
   cd lossgainrsl-x.yy
   ```
4. In the simplest case, where C++ compiler is named `g++` in the system, and parallel computing is not planned, the `make` command without parameters can be enough.
5. Otherwise, the `Makefile` in the current directory should be edited prior to run `make`. It is recommended to preserve the source file and edit a copy of it. The user should specify: `CXX=` , a command to run the C++ compiler for building mpi-enabled programs; and

`CXXFLAGS=` , required parameters for the compiler. Some typical variants are provided in commented out form in the beginning of the `Makefile`.

6. Once the makefile has been edited, run `make` or `make` *`filename`* (if the new file has been saved with a name *`filename`*).
7. After the successful compilation, a new executable file, `lossgainRSL`, will be created in the `./bin` directory. Error messages or warnings should be analyzed and eliminated e.g. by editing the `Makefile` again. Prior to re-run `make`, the command `make clean` should be executed to clear previous results.
8. When the program has been built successfully, run `make test`.
9. After few minutes, the message `Example completed OK` is displayed, which confirms that the program is operable.
10. To test the mpi-enabled version of the program on two CPUs, run the command `make mpitest`. In this case, the result should be checked as described in the last section, item 6.

3 Command line syntax

In a single-CPU operating mode, lossgainRSL should be run by the following command line:

```
lossgainRSL [options] [%name=value...] [config] [outfile]
```

Here, the program name is shown for linux; in Windows use `lossgainrsl` (32-bit version) or `lossgainrsl64nompi` (64-bit version without MPI) or `lossgainrsl64` (64-bit version with Microsoft MPI support). The command line is case-insensitive except for file names in linux. All command line arguments are optional. Short help on the command line syntax and options will be displayed if the program is run with any of the arguments `? -? -h`. The program options are specified by a dash (`-`) or double dash (`--`) followed by a letter and value if necessary (parameters of the gene selection predicate are specified in a different form, see below). The following options are supported by the current version of lossgainRSL, in alphabetical order:

`-b` If this option is specified, all operators xIN of any sort that are present in the predicate will be performed as xINBK operators, i.e., including backward check (see 5.5.2 for detail). And vice versa, `--b` option make the program to perform each xINBK operator of the predicate as xIN operator, i.e., without backward check.

`-c`*N* Normally, when this option is omitted or specified with *N*=0, for each gene or the reference species, the program looks only for *first* homolog of that gene and proceeds to the next gene of the reference species if the homolog is found. If `-c1` is specified, once a homolog is found, the program continue looking for next homolog in the same species and so on, until *all* homologs of the candidate gene are found in each species. The program performs much slowly in this mode and generates bigger output, so it is not recommended to use the option for complex multispecies search.

`-g`*N* Controls the frequency of output during the search of candidate genes. The value of *N* is a step of displaying serial numbers of genes in each genomic sequence under analysis (chromosome, scaffold, contig, etc.). Zero value cancels this output completely. The step of 10 is used by default.

`-i`*N* The option matters only if protein similarity matrices are used at input. It informs the program on optional fields presence in the matrices (see 4.4). If `-i1` is specified, the program assumes each matrix entry to contain fields 5 and 6 with integer values of the rank and back rank (Table 2). If `-i2` is specified, these fields should contain floating numbers from [0, 1] interval. If the option is omitted or *N*=0 is specified, the matrices are processed as if no optional field is present. This option speeds up the program, if the predicate specifies nonzero values of ORTH/BBH parameter (5.5.1).

`-l` If this option is specified, all operators IN (or INBK) of the predicate are treated as LIN (LINBK). It means that in non-reference species, the program looks for witness genes only in the left semi-neighborhood *including* the homolog itself. See also 5.5.2.

`-ll` The same as `-l`, but all operators IN/INBK are treated as LLIN/LLINBK, i.e., the witnesses are searched in the left semi-neighborhood *excluding* the homolog itself.

`-n` Prevent the program from using MPI e.g. in case of incompatible version of MPI libraries installed in the OS. If lossgainRSL fails due to the lack of MPI, first it is recommended to try this option. If the error persists, use the program version compiled without MPI support.

`-q`*N* This option allows shortening the program log output to console. Default *N*=0 is a maximum detailed log. If `-q1` is specified, the program suppresses the gene table error reporting such as a reoccurrence of the protein. If `-q2` is specified, the program does not output a list of species and predicate specified in the configuration file as well as error messages regarding the table of homologs (orthologs or paralogs). If `-q3` is specified, the program in addition does not output details of reading input files and searching genes, and output only final result thus producing the shortest log.

`-r` If this option is specified, all operators IN (or INBK) of the predicate are treated as RIN (RINBK). It means that in non-reference species, the program looks for witness genes only in the right semi-neighborhood *including* the homolog itself. See also 5.5.2.

`-rr` The same as `-r`, but all operators IN/INBK are treated as RRIN/RRINBK, i.e., the witnesses are searched in the right semi-neighborhood *excluding* the homolog itself.

`-x` Outputs only sought-for genes of the reference species along with their homologs in other species even if entire synteny blocks (i.e., including witnesses) were requested in the configuration file. And vice versa, `--x` option instruct the program to output genes of any species together with witnesses in that species irrespective of the configuration setting.

`%`*name=value* The program allows the user to specify numerical parameters in the configuration file (in the form of `%`*name*, where *name* is an identifier); the effective values of the parameters to be specified in the command line by this argument(s). Separate argument is required for each named parameter; the order is arbitrary. Numerical values may be of integer or float types, written in fixed or scientific form. String values should not contain blanks. To use the command in Windows command scripts, the percent character should be doubled.

Above options may appear in any order at any place of the command line. The first positional argument that is neither option nor parameter is used as the name of configuration file (*config*), the second – as the name of output file (*outfile*).

config The name of configuration file including optional path. Default is file `config.ini` in the current directory. The configuration file is described in section 5.

outfile The name of output file including optional path. Default is file `result.txt` in the current directory. The output file is described in section 6.2 of this manual.

The lossgainRSL program outputs its log (6.1) to the standard stream *stdout* (the console by default) which can be redirected to a file, if desired, by appending the command line with the argument >*logfile*. Debug information, if requested in the configuration file, will output to the standard stream *stderr* (the console by default) and can be redirected to a separate file by adding the argument, 2>*debugfile*, to the command line.

The lossgainRSL program (64-bit version only) supports parallelization in the MPI environment. The following command line should be used for multiprocessing:

```
mpirun -np P lossgainRSL [...]
```

where *P* is the number of logical CPUs the program will run on. Here, the program names in linux are shown; Windows users should specify `mpiexec` and `lossgainrsl64` instead. Other arguments of the command line were described above.

4 Input data

Preferred use of the lossgainRSL program relies on input data from the Ensembl database (www.ensembl.org). Doing so, the set of species in a study is naturally limited to those included in the database (as of April 2021, version 106 of the database includes 310 vertebrate genomes). The program ability of working with species that are absent from Ensembl is discussed in chapter 7.

When using input data from Ensembl, the program requires two or three types of input files depending on the configuration file. These files are described in below sections 4.1–4.3. The user can choose from the two ways for obtaining these files:

1) The user submits a proper SQL query to a public Ensembl MySQL server (see http://www.ensembl.org/info/data/mysql.html for details) through a client application. We do not describe this option in detail as it is intended for skilled users who know SQL and have learned the Ensembl database schemes. In addition, a complex SQL query can take much time to process so the public server will forcedly terminate it. In such cases users are recommended to set up a mirror database at their premises, which can be rather difficult.

2) The user enters the BioMart data-mining tool interface at the Ensembl web site (http://www.ensembl.org/biomart/martview/), in order to obtain necessary data interactively. It does not require deep skills but can be tedious. In some cases the files have to be merged

from parts using some ancillary facilities such as OS shell or text editor. It is precisely this method that we describe below in sections 4.1–4.3 for each data type.

As is described in chapter 7, the lossgainRSL program allows using, instead of Ensembl, some other sources such as GenBank and OrthoDB, but the user must compose the necessary files manually or with assistance of their own software. Yet three alternative types of input data supported by the current program version are described in sections 4.4–4.6.

4.1 Table of genes

This table is required for each species involved in the study. It contains IDs of all proteins and genes of this species as well as gene coordinates, names and annotations. The table is a TSV (tab-separated values) formatted text file containing field names in the first line. Each subsequent line describes a protein (for protein-encoding genes) or RNA (transfer, ribosomal etc.). These lines appear in arbitrary order. The name of the table must coincide with a species name in the configuration file; all such tables must be in the same directory and have the same extension(s) of the file name.

The set and order of fields in the tables of genes may differ, but each table must include at least the fields listed in Table 1.

Table 1. Obligatory fields of the table of genes.

Name in Ensembl v106	Description
Protein stable ID	Identifier of the protein in the Ensembl database
Gene stable ID	Identifier of the gene in the Ensembl database
Chromosome/scaffold name	Identifier of the top-level genomic sequence (chromosome, scaffold, contig...)
Gene start (bp)	Starting position of the gene in the sequence
Gene end (bp)	Ending position of the gene in the sequence
Strand	Indication of the strand: 1 is positive (forward), –1 is negative
Gene name*	Symbolic name of the gene (the gene symbol)
Gene description*	Description or annotation of the gene

Note: The fields labeled with *, strictly speaking, are not required by the program, but highly recommended for inclusion in the table of genes.

Field names in the table corresponds to Ensembl version 106. The lossgainRSL program does not assume these names unchangeable, but all tables must use the same name of each field. Moreover, the field names at output may differ from those shown in Table 1; the mapping can be set in the configuration file and does not require modifying the program.

Examples of the table of genes are provided in `genes` directory within the test example (http://lab6.iitp.ru/en/lossgainrsl/example.zip). Let us give the sequence of actions resulting in e.g. `Human.tsv` file in that directory:

1. Enter the BioMart interface (http://www.ensembl.org/biomart/martview/)
2. In the CHOOSE DATABASE list box, select Ensemble Genes 106 (or later).

3. In the CHOOSE DATASET list box, select Human genes (...).
4. Click Filters in the left, then expand REGION in the right.
5. Check Chromosome/scaffold box and select Y chromosome from the list.
6. Click Attributes in the left and select Features in the right, then expand GENE below.
7. Uncheck all fields, then check the fields listed in Table 1. Those fields will appear in each line of the file in the order they were checked.
8. Press Results button in the top left. In the right, select Export all results to File, TSV and check Unique results only, then press Go button.
9. A file named mart_export.txt will be downloaded to the user's directory for downloads. Rename this file to Human.tsv and move it to the directory containing all tables of genes.

4.2 Table of orthologs

This table is required for the reference species as well as for each substitute species participated in a 3-species condition (see 5.5.6 for details). The table is a TSV-formatted text file; heading line is not needed and ignored if appears. Each line of the table contains two fields with Ensembl IDs of two orthologous genes: the first one of the species the table is pertinent to (i.e., the reference or substitute species), and the second gene of some other species.

Notice that the table of orthologs for the reference or substitute species must contain orthologs of *each* gene of this species in *all* other species, therefore it can be voluminous. No order of lines is assumed but can be helpful. The name of the table must coincide with the species name in the configuration file; all such tables must be in the same directory and have the same extension(s) of the file name.

Example of the table of orthologs is provided in `orthologs` directory within the test example (http://lab6.iitp.ru/en/lossgainrsl/example.zip). Let us give the sequence of actions resulting in `Mouse.tsv` file in that directory:

1. Enter the BioMart interface (http://www.ensembl.org/biomart/martview/)
2. In the CHOOSE DATABASE list box, select Ensemble Genes 106 (or later).
3. In the CHOOSE DATASET list box, select Mouse genes (...).
4. Click Filters in the left, then expand REGION in the right.
5. Check Chromosome/scaffold box and select Y chromosome from the list, then shrink REGION panel.
6. Click Attributes in the left and select Homologues in the right, then expand GENE below.
7. Leave checked only Gene stable ID field, then shrink GENE panel.
8. Expand ORTOLOGUES [...] panel below and scroll it to a species to look for orthologs in, e.g. Human. (Among several panels, the user should expand one containing a letter the species common name begins with, here [F-J].) Check Human gene stable ID box.
9. Click Filters in the left, then shrink REGION and expand MULTI SPECIES COMPARISONS. Check Homologue filters box, choose Only, then select Orthologous Human Genes from the list box.
10. Press Results button in the top left. In the right, select Export all results to File, TSV and check Unique results only, then press Go button.
11. A file named mart_export.txt will be downloaded to the user's directory for downloads. Rename this file to Mouse_Human.txt.
12. Click Attributes in the left and uncheck the box previously checked in item 8. Then repeat items 8-12 with another species until all "second" species are examined for given "first" one.

13. Merge all files obtained in item 11. For example, the following Windows command will do:
    ```
    copy /a Mouse_*.txt /b Mouse.tsv
    ```
14. Move the result file, Mouse.tsv, to the directory containing all tables of orthologs.

4.3 Table of paralogs

This table is needed if orthologous genes should be selected from not only orthologs but also paralogs of them as specified in the configuration file. Another case is when the number of paralogs has to be output for selected genes (the field named #, see 5.3). The table is required for each species whose paralogs should be considered. It is a TSV-formatted text file; heading line is not needed and ignored if appears. Each line of the table contains two fields with Ensembl IDs of two paralogous genes of the same species this table is pertinent to. The table must contain all *ordered* (i.e. including transpositions) pairs of paralogous genes of given species; the order of pairs does not matter. The name of the file must coincide with a species name in the configuration file; all such tables must be in the same directory and have the same extension(s) of the file name.

Example of the table of paralogs is provided in `paralogs` directory within the test example (http://lab6.iitp.ru/en/lossgainrsl/example.zip). Let us give the sequence of actions resulting in `Mouse.tsv` file in that directory:

1. Enter the BioMart interface (http://www.ensembl.org/biomart/martview/)
2. In the CHOOSE DATABASE list box, select Ensemble Genes 106 (or later).
3. In the CHOOSE DATASET list box, select Mouse genes (...).
4. Click Filters in the left, then expand REGION in the right.
5. Check Chromosome/scaffold box and select Y chromosome from the list.
6. Shrink REGION in the right, then expand MULTI SPECIES COMPARISONS. Check Homologue filters box, choose Only, then select Paralogous Mouse Genes from the list box.
7. Click Attributes in the left and select Homologues in the right, then expand GENE below.
8. Leave checked only Gene stable ID field, then shrink GENE panel.
9. Ensure that below Attributes in the left the only field is Gene stable ID, otherwise uncheck extra fields under GENE or ORTHOLOGUES in the right. Then expand PARALOGUES in the right and check Mouse paralogue gene stable ID box.
10. Press Results button in the top left. In the right, select Export all results to File, TSV and check Unique results only, then press Go button.
11. A file named mart_export.txt will be downloaded to the user's directory for downloads. Rename this file to Mouse.tsv and move it to the directory containing all tables of paralogs.

4.4 Protein similarity matrix

This data type is an alternative for the tables of orthologs and paralogs from Ensembl (4.2, 4.3). Notice that the table of genes (4.1) is still required for each species involved in the study though in this case such table may contain protein and gene IDs not from Ensembl.

The protein similarity matrix is required for each species. It is a TSV-formatted text file without heading line, which is ignored if appears. Each data line of the matrix file should contain obligatory fields 1–4 and optional fields 5–6 listed in Table 2. The name of the file must coincide with the species name in the configuration file; all matrices must be in the same directory and have the same extension(s) of the file name.

The matrix lines can appear in an arbitrary order, but the values of fields 3 and 4 must be consistent over all matrices. Thus, if the matrices are formed using BLASTP, the same parameters have to be used everywhere, including the amino acid substitution table such as BLOSUM62. The entries may be non-unique: multiple occurrence of the same protein pair is allowed, and lossgainRSL will use one corresponding to the maximum raw score and minimum E-value.

Table 2. Fields of the protein similarity matrix entry.

No.	Description
1	A protein identifier of the species corresponding to the matrix name. If all tables of genes comply with section 4.1, the Ensembl protein ID has to be used as the identifier. If a table of genes has compatible format but relies on other protein identifiers, those ID have to be used in the protein similarity matrix. If a line of the matrix contains unknown protein ID, such line will be skipped with a warning. **Note:** The program will aggregate these data to form the matrix of genes by choosing, for each pair of genes, a pair of their proteins with maximum similarity. Therefore, this field may (and should) directly contain gene ID.
2	A protein identifier of the same or other species. Similar to field 1 notes apply.
3	The raw score amount for optimal local alignment of the two amino acid sequences for the proteins specified in the fields 1 and 2, or standardized weight calculated from the score.
4	The expectation (E-value) for such alignment.
5	(Optional) The rank of the entry, which is a serial number of this hit of given "first" species protein against the "second" species in descending order of the protein alignment quality. In other words, having the "second" species and a protein of the "first" species fixed, this field contains 1 for best hit, 2 for one next to best, and so on.
6	(Optional) The back rank of the entry, which is a serial number of this hit of given "second" species protein against the "first" species in descending order of the protein alignment quality. In other words, having the "first" species and a protein of the "second" species fixed, this field contains 1 for best backward hit, 2 for one next to best, and so on. Particularly, if the proteins are BBH, both fields 5 and 6 contain 1. This field may present only in presence of field 5. If there is no backward hit, the field may be omitted or contain the special number of 999. **Note.** Described contents of the fields 5 and 6 assume the program running with `-i1` option. When using `-i2` mode, both fields should contain real numbers between 0 (best hit) and 1 (absent hit).

An opening portion of the protein similarity matrix imported to Excel is exemplified in the 'Xenopus_scores' sheet of the file http://lab6.iitp.ru/en/lossgainrsl/dataformat.xlsx. Such matrix can be formed with BLASTP, provided that amino acid sequences for all species and protein IDs have been prefetched from the Ensembl or another source. Another example of the matrix is provided in the 'Zebrafish_weights' sheet of the same file. Here, the fields 1 and 2 contain NCBI gene IDs, field 3 is the hit weight, normally between 0 and 1, and field 5 is present, which is a rank of the hit.

4.5 Table of orthologous groups

This data type is concentrated in a single table, which substitutes for all data normally obtained from Ensembl as described in 4.1–4.3. The table is a TSV-formatted text file generated by the user's software from the available data, in particular, OrthoDB (https://www.orthodb.org/). The table may be physically split into several files as listed in the configuration, which shall be logically merged by the lossgainRSL program.

First line of the table must contain the field names for all subsequent lines excepting empty ones, which separates the groups. The heading line allows any order and set of fields, but they must include at least the fields listed in Table 1 with the only exception: instead of the ‘Protein stable ID’ field (which contains the protein identifier), the ‘Species’ field is required that contains the identifier of a species from the configuration file. The names and order of fields may differ from those shown in Table 1; actual names have to be respectively specified in the configuration file. Here, gene and sequence IDs are taken from the source used to build the table of orthogroups.

Let us stress that here *one* table contains the information on genes of *all* species participating in the study. The arrangement of lines in the table is an important part. Each line describes a gene of some species, and such lines are grouped in *orthogroups* composed of pairwise orthologous genes from same or different species. The orthogroups are separated from one another with empty line. The order of lines within an orthogroup does not matter. The genes that have no orthologs are omitted from the table.

A portion of the table of orthogroups imported to Excel is exemplified in the ‘Orthogroups’ sheet of the file http://lab6.iitp.ru/en/lossgainrsl/dataformat.xlsx.

4.6 Table of protein clusters

This data type is concentrated in a single table, which substitutes for orthology/paralogy data normally obtained from Ensembl (sections 4.2, 4.3). However, the table of genes (4.1) is still required for each species involved in the study. The table is a TSV-formatted text file generated with use of previously developed program for protein clustering (http://lab6.iitp.ru/en/pr_protclust/). The table may be physically split into several files as listed in the configuration, which shall be logically merged by the lossgainRSL program.

First line of the table must contain the field names for all subsequent lines excepting empty ones, which separates the clusters. The heading line allows any order and set of fields, but they must include the field with gene or protein identifier from those listed in Table 1.

Let us stress that here *one* table contains the information on proteins or genes of *all* species participating in the study. The order of lines in the table is an important part. Each line describes a protein or gene of some species, and these lines are grouped in *clusters*. The cluster is composed of pairwise orthologous genes/proteins of the same or different species. The clusters are separated by empty line; the order of lines within a cluster does not matter. The orthology of genes is determined by the splice variant that result in proteins of greater likeness. Thus, each gene occurs in the table only once or never, since clusters composed of one gene (singletons) are omitted from the table.

A portion of the table of protein clusters imported to Excel is exemplified in the ‘Gene_clusters’ sheet of the file http://lab6.iitp.ru/en/lossgainrsl/dataformat.xlsx.

5 Configuration file

The configuration file is used to define a task in hand and to control the lossgainRSL operation. The file is crucial as in most cases the program operation fails due to erroneous configuration. The configuration file is required to run the program; a name of this file must be specified in the command line or coincide with the default name (ref. to 3).

The file should be created in usual text format, but line breaks are not allowed. If a line of the file contain several fields, they should be separated by at least one tabulation character. A dot, comma and blank characters are not delimiters; they can occur inside the fields as desired. The configuration file may include comments, which are ignored. The entire line of comments should begin with semicolon (;) or double slash (//). Also ignored is the line part that starts with double slash.

For the lossgainRSL program use, the configuration file must include the five sections which are described below in the order they should appear in the file. Extra sections may present but will be ignored. Each section begins with a heading line containing the section name in brackets. The section data lines follow until a new section heading or end of file is met. Several configuration file variants are provided in the test example (http://lab6.iitp.ru/en/lossgainrsl/example.zip).

5.1 Program operation modes: [mode]

In most cases this section consists of a single data line containing one field. The field should contain in arbitrary order some of uppercase letters explained in Table 3. Not all letters may co-occur; each group of mutually exclusive letters is shown between horizontal rules. Characters that are absent from the table, such as blank, comma, tabulation, etc. will be ignored; one can use them for legibility.

Table 3. lossgainRSL mode characters.

Letter	Description
	Variant of input data – one option must be chosen
H	Principal mode of the program. Input data from Ensembl are used: the tables of genes (4.1), orthologs (4.2) and paralogs (4.3) as needed.
A, W	The tables of genes (4.1) and protein similarity matrices (4.4) are used as input data. In A mode, each matrix is forcedly symmetrized: in each pair of symmetric elements of the matrix, better hit substitutes worse one. In W mode, the matrix is used "as is", whether symmetric or not. First option is much slower, so we recommend to symmetrize the matrices externally at the point of creation, rather than widely use mode A.
Q	The only input data is the table of orthogroups (4.5).
C	The tables of genes (4.1) and united table of protein clusters (4.6) are used as input data.
	Variant of the distance between genes in a neighborhood (Fig. 1)
E	Specified radius of neighborhood will be compared to the distance between most distant ends of the genes (default mode).
B	The radius of neighborhood will be compared to the distance between the gene middles.
S	The radius of neighborhood will be compared to the intergenic distance, i.e. the distance between proximate ends of the genes (negative if the genes overlap).
	Synteny extra options* – any combination thereof (Fig. 2)
O	Mutually orthologous genes of two synteny blocks must occur in the same order ignoring their orientation.

Letter	Description
D	Mutually orthologous genes of two synteny blocks must have the same direction, but their order does not matter.
M	Synteny conditions also include the mirror pattern, where both orientation and order of all genes are inverted.
	Presentation of output data
X	In this mode, for each synteny block only sought-for gene of the reference species and its homologs in other species will appear in the output file as single line. No witness data are printed. Otherwise, the entire synteny block will be output as several lines; but first line always contain the data as in X mode. Witness data are printed in following lines. This setting can be changed in the command line, see chapter 3.
I	In X mode, this option is ignored. If X was not specified, synteny blocks will be separated by empty line in the output file.
J	In X mode, this option is ignored. If X was not specified, two fields are appended to each line of the output file, including empty line due to I option. First field contains ID of the sought-for gene pertinent to this synteny block, second field is for a line number in the block. Thus, if a synteny block in the reference species consists of gene *Id* and two witnesses, such block will be presented as four lines in the output file: for *Id*, first witness, second witness, and empty line. These line contain the following data pairs in the two appended fields: (*Id*, 0), (*Id*, 1), (*Id*, 2) and (*Id*, 3) respectively. This allow sorting the output file by genes found e.g. in the *Id* ascending order.
	Debug mode – ignored in MPI parallel environment
G	Declares that additional lines of the [mode] section contain IDs of one or more genes of the reference species, and those genes processing must be detailed in the debug log. Such log will be output to the standard stream, *stderr* (ref. to chapter 3).
P	The same, but additional lines contain protein IDs of the reference species. The processing of genes encoding such proteins will be detailed in the debug log.
T	The same, but additional lines contain names of sequences (chromosomes, contigs, etc.) in the reference species genome. The debug log will detail the processing of each gene of a sequence.
N	The same, but additional lines contain serial numbers of sequences in the reference genome with optional number of a gene (separated by underscore). All numbers are zero based. For instance, 493_5 means sixth gene in the 494th sequence of the reference genome.
V	This option can be used together with any other from this group; it enables the verbose debug output. However, it can result in a log of huge size. It is not recommend to specify more than one gene in this mode.

Note:

* This group options are not mutually exclusive; they may appear in any combination or absent at all. The latter means that the only condition of synteny is the presence of witnesses in the specified neighborhood irrespective of the order and orientation of these genes.

After the line of mode letters, one or more data lines can follow. These lines specify objects to obtain debug information for. Such objects may be IDs or serial numbers of proteins, genes or genomic sequences. Each data line contain one or more objects separated by tab characters. These data are processed in accordance with specified debug mode from Table 3; invalid IDs or numbers are ignored. If no debug mode was specified, all debug lines are ignored. More information on debugging is provided in section 5.5.7.

The following figures illustrate variants of the distance between genes and synteny extra options currently implemented in the lossgainRSL program.

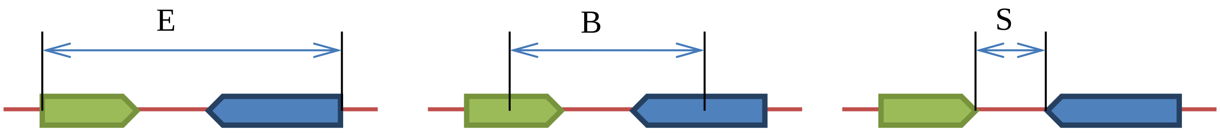

Fig. 1. Variants of distance for the proximity checking.

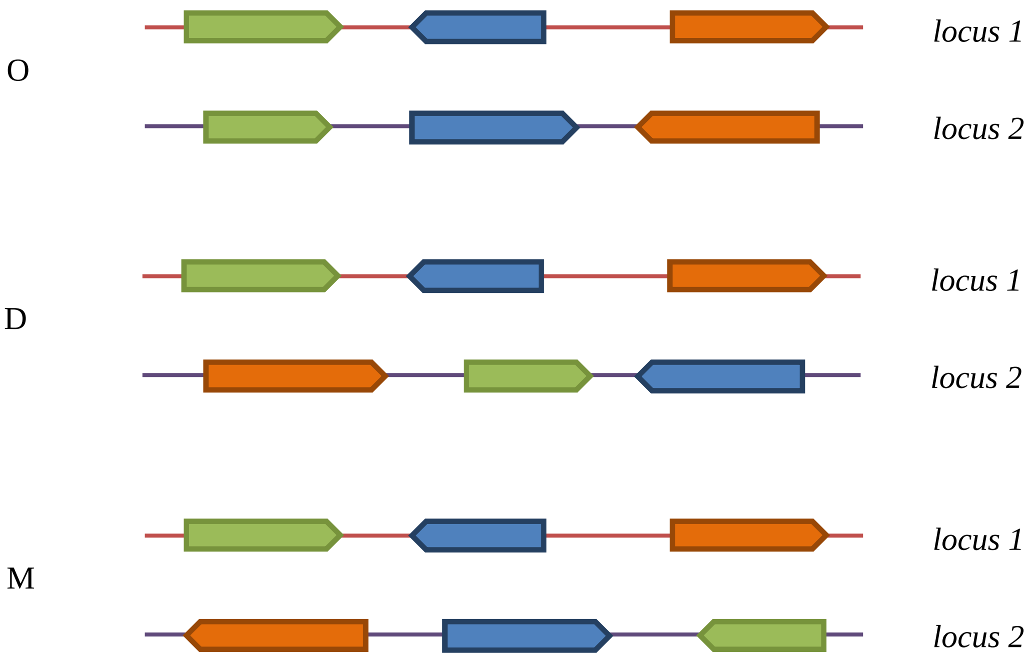

Fig. 2. Synteny extra options for two loci.
O: same order of genes, D: same orientation of genes, M: allow mirror arrangement.

5.2 Input data location: [data]

This section of the configuration file is to specify directories, where each type of input data is stored as well as corresponding file names and extensions. In most cases, a file name (or variable part of it) coincides with a species name from those listed in the [species] section, see 5.4. Thus, the wildcard (*) is normally used for file names in this section.

This section consists of several lines in the same format. They can appear in arbitrary order. Each line describes one of input data types and contains two fields separated by at least one tab character. First field of the line contains an uppercase letter as per data type, respective second field is described in Table 4. The program uses only appropriate data types for the mode selected and ignore extra lines. Therefore, this section of the configuration file can follow a generic pattern, e.g. in Windows:

```
[data]
I     genes\*.tsv
O     orthologs\*.tsv
P     paralogs\*.tsv
H     homologs\*.tsv
A     scores\*.tsv
Q     orthogroups\odb10.tsv
C     protein-clusters.tsv
```

Table 4. Input data types.

Type	Description of the second field
I	An absolute or relative path to the directory containing the tables of genes (ref. to 4.1). The wildcard * should be specified as a file name followed by the extension(s) if any. Thus, all tables of genes must localize in this directory and have the same extension(s).
O	An absolute or relative path to the directory containing the tables of orthologs (ref. to 4.2). The wildcard * should be specified as a file name followed by the extension(s) if any. Thus, all tables of orthologs must localize in this directory and have the same extension(s).
P	An absolute or relative path to the directory containing the tables of paralogs (ref. to 4.3). The wildcard * should be specified as a file name followed by the extension(s) if any. Thus, all tables of paralogs must localize in this directory and have the same extension(s).
H	An absolute or relative path to the directory containing joint tables of orthologs and paralogs for all or some species. In fact, such table is a union of the two tables described in sections 4.2 and 4.3. The wildcard * should be specified as a file name followed by the extension(s) if any. Thus, all tables of homologs must localize in this directory and have the same extension(s).
A	An absolute or relative path to the directory containing the protein similarity matrices (ref. to 4.4). The wildcard * should be specified as a file name followed by the extension(s) if any. Thus, all protein similarity matrices must localize in this directory and have the same extension(s).
Q	An absolute or relative path and name of the file containing the table of orthogroups (ref. to 4.5). If the table was split into several parts, multiple lines with different file names should be specified in this section. These parts will be merged in the order specified. The head line with field names is required only in the first part, but may also be present in other parts.
C	An absolute or relative path and name of the file containing the table of protein clusters (ref. to 4.6). If the table was split into several parts, multiple lines with different file names should be specified in this section. These parts will be merged in the order specified. The head line with field names is required only in the first part, but may also be present in other parts.

5.3 Relevant field names: [fields]

The table of genes (4.1), orthogroups (4.5), and protein clusters (4.6) can contain many fields in each entry and have rather free format: both list and order of fields are unrestricted, which simplify the preparation of input data. However, there are several fields analyzed by the algorithm or transferred to output data. So we require that these files always contain the names of *all* fields in the head (first) line, and this section of the configuration file has to specify what field name corresponds to a certain data type. In addition, it is possible to specify optional alias of a field for convenience.

This section consists of six or more lines. Each line contains one or two tab-separated fields. First field is the name of a data field in the head line, and second field defines optional alias. These six lines must appear in the following strict order:

1) *Protein/Species*. If the gene tables (4.1) with or without protein clusters (4.6) is used, it is a name of the input field that contains a protein identifier in the Ensembl or other source. Otherwise, if the table of orthogroups (4.5) is used, it is a *species* identifier (see 5.4).
2) *Gene*. This is a name of the input field that contains a gene identifier in the Ensembl or other source.
3) *Contig*. This is a name of the input field that contains the identifier of a top level sequence (chromosome, scaffold, contig, etc.) in the Ensembl or other source.
4) *Start*. This is a name of the input field that contains first position of a gene in the respective DNA sequence. (The first position is always less than the last one irrespective of strand.)
5) *End*. This is a name of the input field that contains last position of a gene in the respective DNA sequence. (The last position is always greater than the first one.)
6) *Strand*. This is a name of the field that contains 1 if the gene occurs in a positive (forward) strand, and –1 for negative (complement) strand. Otherwise, the characters '+' and '–' can be used.

These six fields are required for the lossgainRSL program to operate. Additional lines may follow to describe other necessary fields such as gene name, annotation and so on. For input data field to appear in output file, such field must be described in this configuration section. Specifically, if the gene tables are obtained from Ensembl v106, this section can look as follows:

```
[fields]
Protein stable ID          Protein
Gene stable ID             Gene
Chromosome/scaffold name   Contig
Gene start (bp)            Start
Gene end (bp)              End
Strand                     +|-
Gene name                  Name
Gene description           Description
```

In addition, there is a special field with dedicated name, #, which can also be specified. This field does not exist in input files, but can be virtually created by the program and appear at output. The field contains the number of paralogs of the gene, for which it is printed. The field # must be specified the very last, both in this section and at output in the [predicate] section, ref. to 5.5.4.

5.4 Species and group names: [species]

This section of the configuration file is intended to list names of all species involved and to define groups uniting some of these species as appropriate. Any unique identifier may be used as a species name, if it satisfies OS requirements for a file name. We recommend to use only alphanumeric characters in the names, substituting underscores for blanks. The use of uppercase/lowercase characters must be consistent throughout the configuration file even if the file system does not distinguish them. Each species must occupy the separate line of this section. The first species is always considered as the reference one, subsequent species may appear in arbitrary order.

Sometimes conditions for the selection of sought-for genes can be formulated more concisely using a notion of *species group* which unite several species to be tested identically. Such group can be established on the basis of any trait, not only a taxonomic one. The syntax of this section does not allow a species to belong to several groups; as a workaround the user can repeatedly include such species name in the list.

The name of a group always begins with asterisk (*) to distinguish it from species names. To define a group, its name should be specified in a separate line among the names of species. The group will be composed of the species listed from the next line on, until the name of another group or end of the section. Examples of this section can be found in the configuration files of the test example (http://lab6.iitp.ru/en/lossgainrsl/example.zip).

5.5 Gene selection predicate: [predicate]

This section of the configuration file is the most both complex and important one, because it is the predicate what determines selected genes and output data. The section uses a special procedural language which is similar to a programming language having some peculiarities and a limited set of functions. The user should keep in mind that he/she is fully responsible for the predicate correctness, while lossgainRSL performs only superficial checking. From our experience, in most cases the program failure and misoperation result from mistaken predicate.

The predicate definition language allows user to specify rather complex conditions for gene selection and easily adjust those conditions with no change of the program. However, the language does not pretend to applicability in *all* situations and does not allow formulating *arbitrary complex* conditions either. This language just offers a number of typical elements to be checked, and methods of combining such elements in a complex test. The user should decide on suitability of these facilities for the task in hand and draw a suitable predicate himself. Sometimes the solution can be obtained by running the program several times with different predicates and then building the sought-for set of genes as a union or intersection of the sets resulted from certain runs.

The program lossgainRSL uses the predicate as follows. With each next gene of the reference species, the predicate is interpreted from the very beginning with no regard to previous gene processing history. The first operator, SET, sets the parameter values which will be used for the current gene of the reference species as well as for other species unless otherwise specified. For those non-reference species, the parameters can be changed by another SET operator(s) or a variant thereof, but such change is in effect only until a new change or end of current gene processing.

The initial SET of the predicate is followed by a series of testing user-specified conditions. Each condition is related to genes of either the reference species and another species (*2-species* condition) or the reference species, so-called substitute species and another species (*3-species* condition). The conditions are tested as they appear in the predicate, but this order of execution can be modified using unconditional and conditional (depending on result of the last test) branches. To this end, *labels* are used including two predefined labels, YES and NO, which mean that the current gene of the reference species satisfies the predicate (the gene is selected) or does not satisfy the predicate (the gene is skipped), respectively. The ADD operator allows the user to specify, for a gene selected in any species, its data field to be copied to output file.

The sequential interpretation of the predicate normally yields one of the two results: (1) the program reaches the YES label or the predicate end (implicitly followed by YES label), therefore the current gene of the reference species is selected and present in the output file; or (2) the program reaches the NO label, therefore the current gene is skipped and absent from the output file. In either case the program proceeds to the predicate evaluation with the next gene of the reference species. After all genes of the reference species have been tried, lossgainRSL finishes.

The following subsections formally describe all means and capabilities of the predicate definition language. Examples of complete predicates can be found in the test example configuration files (http://lab6.iitp.ru/en/lossgainrsl/example.zip).

5.5.1 Operator SET and its variants

This operator is intended for setting or changing parameters of the sought-for gene selection. Like other operators, it must be entirely written in a single line of the configuration file. The operator contains from 1 to 11 fields separated by the tab character(s). The syntax is:

```
SET[,w,r,b,k,s,e,l,h1,h2] [WIT[NESS]=w] [RANGE=r] [{ORTH|BBH}=b] [RANK=k]
[SCORE=s] [EVAL=e] [LAMBDA=l] [HOLD1=h1] [HOLD2=h2] [HEAD[ING]=CS-list]
```

There are nine numerical parameters which can be set in two ways. First, comma-separated values of the parameters can be specified in the first field in fixed order just follow the operator code. Unchanged parameters may be skipped still preserving commas if other values follow. Second, the parameters can be set in any number and order in the subsequent fields of the operator; each parameter is recognized by the keyword assigned to it. The parameter description, allowable and default values as well as the keyword are given in Table 5. Also included is optional text parameter of the program which cannot be set in the first field of the operator, but only in a separate field (or operator) with the keyword HEAD or HEADING. This parameter defines the head line of the program output file.

The value of a numerical parameter can be specified as an integer/real number or a reference in the form %*name*, where *name* is an identifier different from other keywords. In the latter case, the parameter value will be specified in the command line as described in chapter 3. Similar way may be used to define string-type parameters for the operator HEAD, but such strings should not contain blanks.

Table 5. The parameters changeable by SET operator.

Symbol	Keyword(s)	Values	Default	Description
w	WIT WITNESS	0, 1, 2	2	Number of witness genes in a synteny block besides the current gene *X*.

Symbol	Keyword(s)	Values	Default	Description
r	RANGE	> 0	2000000	The half-size of a neighborhood, where witnesses of the current gene are looked for (see also Fig. 1). The value of *r* is rounded to the nearest integer; it may include the suffix K|k or M|m for values in kilobases or megabases, respectively. Thus, a neighborhood with the half-size of 200 kbp can be specified as 200000, 200K or 0.2M.
b	ORTH BBH	0...511	0	1) For input data from Ensembl, values 0...7 (i.e., three bits) are allowed, where each bit controls the way an orthologous gene is defined: 1 means the ortholog, and 0 – the ortholog or any of its paralogs. Bit 0 (least significant bit, LSB) corresponds to the current gene *X*, bit 1 – to first witness *Y*, bit 2 – to second witness *Z*. For instance, *b*=1 means that only orthologs are selected for *X*, and orthologs or their paralogs for *Y* and *Z*; *b*=7 means that only orthologs of *X*, *Y*, and *Z* are considered. 2) If the protein similarity matrices are used at input (see 4.4), this parameter specifies requirements for homologs in terms of the raw score magnitude. The meanings of individual bits are as follows: 1 – BBH (bidirectional best hit) is taken for gene *X*, 2 – BBH is taken for gene *Y*, 4 – BBH is taken for gene *Z*, 8 – BHF (best hit forward) is taken for gene *X*, 16 – BHF is taken for gene *Y*, 32 – BHF is taken for gene *Z*, 64 – BHB (best hit backward) is taken for gene *X*, 128 – BHB is taken for gene *Y*, 256 – BHB is taken for gene *Z*. Notice that if both BHF and BHB are specified for a gene, respective BBH bit is set automatically.
k	RANK	> 0	10	This parameter is actual only if input data include the protein similarity matrices (4.4). The specified value is a cut-off: only first *k* hits in descending order are considered as homologous proteins when the program is running with option `-i1`. If option `-i2` is used, *k* should be a real number and all hits with less values of the rank from [0, 1] are considered (see 3).
s	SCORE	≥ 0	0.001	This parameter is actual only if input data include the protein similarity matrices (4.4). The specified value is a cut-off: two proteins are considered homologous if local alignment of them yields raw score or weight that is greater or equal to such cut-off.

Symbol	Keyword(s)	Values	Default	Description
e	EVAL	≥ 0	1e-5	This parameter is actual only if input data include the protein similarity matrices (4.4). The specified value is a cut-off: two proteins are considered homologous if local alignment of them yields E-value that is less or equal to such cut-off.
l	LAMBDA LAMB	0–1	0	This parameter is actual only if input data include the protein similarity matrices (4.4). The specified value will be used for λ in a backward check condition (see 5.5.2).
h1	HOLD1	0...7	1	The purpose of a 3-bit value of this parameter is to instruct the program, for the current gene of the reference species, which genes vary during the predicate evaluation and which ones are invariable. The meanings of individual bits are as follows: 1 – the test must be done with the same gene *X*, 2 – the test must be done with the same witness *Y* (otherwise the predicate is not satisfied), 4 – the test must be done with the same witness *Z*. For example, if the value of 7 is specified and the predicate includes two elementary conditions, both must be satisfied with the same triplet of genes.
h2	HOLD2	0...7	1	This parameter affects only 3-species conditions (5.5.6); it is similar to *h1*, but controls the relationship between two elementary conditions behind the 3-species condition. The value of 1 should be specified in case of input data from the Ensembl.
–	HEAD HEADING	The value of this parameter is a text string that specifies comma-separated headings of the fields in the output file, left to right. Thus, comma should not be used in a heading text. If the parameter is omitted, field headings in the output file equal to those at input or their aliases as specified in the [fields] section of the configuration file (ref. to 5.3). String-type parameters in the form %*name* may be used to form the string provided that their values were specified in the command line.		

SET must be the first operator and can also appear later in the predicate in order to modify some parameter(s). Doing so, one should take account of: (a) modified parameter values will be in effect until the next alternation or end of predicate; (b) the size of neighborhood in the reference species cannot be modified dynamically; the value specified in the initial SET operator is always used, and modified *r* will apply only to non-reference species.

If not all but only few parameters of the lossgainRSL are to be modified, it can be more convenient to specify new values not following the operator code, but in subsequent field(s) with use of a corresponding keyword from the Table 5, for example:

```
SET  RANGE=1M  WITNESS=1
```

There are also variants of the SET operator, where the keyword is used as the operator code (first field) and the value is specified in second field, e.g.:

```
RANGE      500K
WITNESS    1
```

Besides the SET operator and its variants, for brevity parameters can be set directly in operators that verify conditions (such as IN, OVER, IN2), similar to the first field of SET. Such local modification is applied only to the operator, where it is specified.

5.5.2 Operators IN and INBK

This operator verifies the presence of current gene *X* of the reference species *R* in other species *A* or a group thereof. This check is successful if the two sub-conditions are simultaneously satisfied: (1) the species *A* includes gene *X'* that is a homolog of *X* in the sense of active parameter settings, e.g. an ortholog of *X*; and (2) there exist specified number of distinct witness genes in the *r*-sized neighborhood of *X*, e.g. *Y* and *Z*, which respective homologs, *Y'* and *Z'*, co-localize in the *r'*-sized neighborhood of *X'* in *A* (Fig. 3). The size of a neighborhood is treated as per chosen variant of the distance between genes (ref. to Table 3 in 5.1), in Fig. 3 it is the ‘E’ variant. Such condition involves two species, therefore it is referred to as *2-species condition*. As the arrows show, homology is verified in the direction from *R* to *A*, which is referred to as *forward check*. Homology requirements for gene *X* and for its witnesses may differ: for example, a witness can require more stringent homology conditions than the gene under check, which is depicted by solid and dashed arrows, respectively.

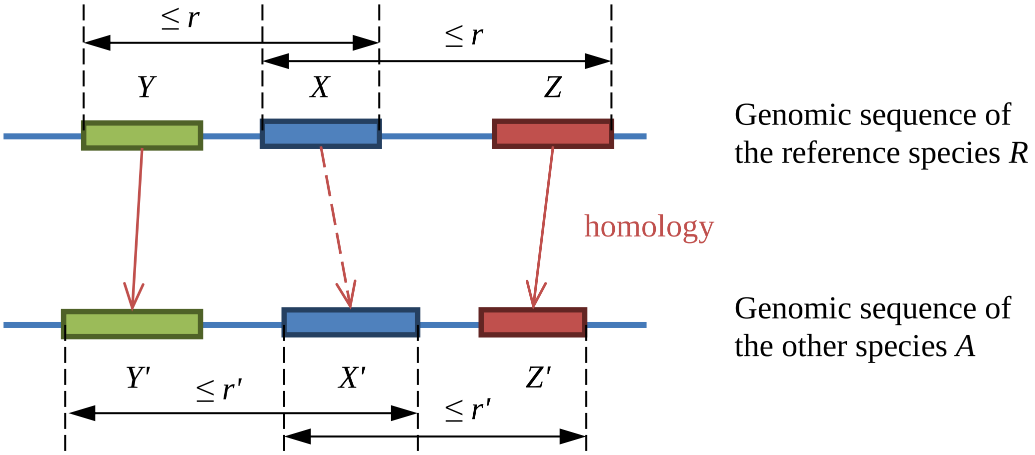
≤ r
≤ r
Y
X
Z
Genomic sequence of the reference species R
homology
Genomic sequence of the other species A
Y'
≤ r'
X'
Z'
≤ r'

Fig. 3. The forward check of 2-species condition for a certain variant of synteny.

If protein similarity matrices (4.4) are used at input, the 2-species condition may include a *backward check* with use of INBK form of the operator IN (Fig. 4). The backward check is applied to homolog *X'* resulting from the forward check; it verifies that there is no alternative *U* for *X* in *R* such that *U* is substantially better, than *X*, homolog of *X'*. With this purpose the program checks the sub-condition (3): there is no gene *U* in *R* such that *X'* is a homolog of *U* with the same witness requirements as for *X* and the following inequality is satisfied:

$$\left[\begin{array}{ll} W(X',U)/W(X,X')>\lambda & \text{if } W(\bullet) \text{ is a raw score} \\ W(X',U)-W(X,X')>\lambda & \text{if } W(\bullet) \text{ is a weight,} \end{array}\right.$$

where $W(\bullet)$ is the value of field 3 of the protein similarity matrix (Table 2), and λ is the parameter *l* in the operator SET (5.5.1).

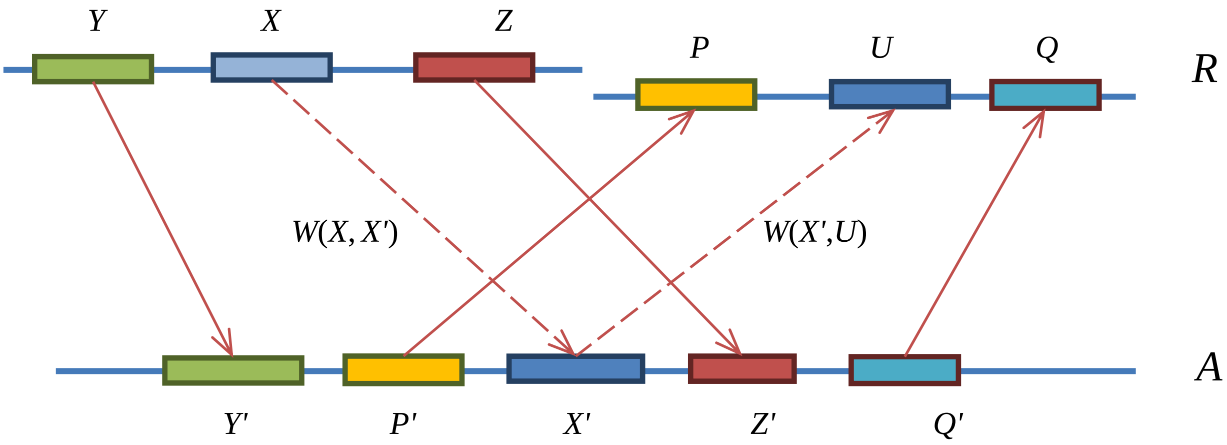

Fig. 4. The 2-species condition with forward and backward checks.

The syntax of IN/INBK operator includes up to three fields:

IN[BK][,*w*,*r*,*b*,*k*,*s*,*e*,*l*,*h1*,*h2*]   *species*   [*yes-label*|–*no-label*]

Here, the first field of the operator allows modifying current parameter values just the same as in the first field of the SET operator, in order to use new values only in this check, not later. The value of *r* is interpreted in a special way: if specified, it determines the size of neighborhood only in non-reference species *A*, i.e. it is *r′* in Fig. 3, while for the neighborhood size *r* in the reference species *R* the program always uses a value specified in the initial SET operator or set by default.

In the second field of the operator, instead of *species* the name of non-reference species *A* or a species group should be specified, i.e., one of the names listed in the [species] section of the configuration file (5.4). Recall that the name of a species group always starts with asterisk (*). If a group has been specified, all species from it are tried in turn until the condition is satisfied; in such case the condition for entire group is satisfied and remaining species are not tried. Otherwise the condition for the species group is unsatisfied.

Optional third field of the IN/INBK operator is a label to go to if the condition is (un)satisfied. Specifically, if a minus sign does not precede the label (*yes-label*) then the program proceeds to this label when the condition is satisfied, otherwise it proceeds to the next operator. Conversely, if the minus sign does precede the label (–*no-label*) then the program proceeds to the next operator when the condition is satisfied, and proceeds to the label otherwise. If this field is omitted, the program proceeds to the next operator in both cases. Further information on labels is provided in the next section.

In fact, the IN operator defines a *multiply nested loop* to verify the condition first over species from the group, then over genomic sequences of each species, then over genes on the sequence, then over homologs of the gene in the other species, then over putative witnesses and their homologs, etc. The depth of nesting is even greater for INBK version of the operator due to checking of putative genes *U* and witnesses thereof. The verification continues until a homolog *X′* and witnesses are found that satisfy the condition or all variants were tried. In the former case the check was successful and the program proceeds to the *yes-label* (if specified), in the latter – unsuccessful and the program proceeds to the *no-label* (if specified). Notice that the loop does not contain any inner operators and consists of a single IN/INBK. Any subsequent actions are performed only after completion or termination of the loop.

There are also variants of the IN operator that allow searching of witnesses in not full neighborhood of the homolog *X′* in *A*, but only in the left or right semi-neighborhood including or excluding the span of *X′* itself. The syntax of these operators is similar to that of the IN operator but the keyword is prefixed as follows:
LIN – the condition (2) is satisfied for the left (at lesser positions) semi-neighborhood of gene *X′* in *A including* the span of *X′*;
LLIN – the condition (2) is satisfied for the left semi-neighborhood of gene *X′* in *A excluding* the span of *X′*;
RIN – the condition (2) is satisfied for the right (at greater positions) semi-neighborhood of gene *X′* in *A including* the span of *X′*;
RRIN – the condition (2) is satisfied for the right semi-neighborhood of gene *X′* in *A excluding* the span of *X′*.
These variants differ from the IN operator only in the domain of genome *A* where witnesses are looked for. The operators also allow the backward checking by addition the suffix ‘BK’, e.g. LLINBK. However, the witness search in a semi-neighborhood is limited to the forward check only, and the backward check is accomplished for full neighborhoods of the genes *X′* and *U*, because the backward check is aimed at prevention of wrong homolog *X′* of current gene *X*.

5.5.3 Labels and operator GOTO

Labels within the predicate serve to change the normal sequence of operators execution. Any unique identifier can be used as a label provided that it does not coincide with a species name, field name or operator code. The labels are case sensitive. Two labels, YES and NO, are defined implicitly (ref. to section 5.5). An explicitly defined label should follow a colon in the separate line, for example:

```
:RareCase
```

The branch to the `RareCase` label means that the next executed operator will be one specified in the line following the label.

The conditional operator IN (5.5.2) is a typical operator that uses a label. Other conditional branch operators are described below which also use labels.

To define predicates having a complex branched structure, the *unconditional branch* GOTO operator is provided. It is specified in a separate line with the following syntax:

`GOTO` *label*

where one of existing labels appears in the second field, for example:

```
GOTO      RareCase
GOTO      YES
```

5.5.4 Operator ADD

This operator helps to form the output file of the desired structure and contents. It allows the user to copy necessary fields from a certain line of the table of genes, for the given species, to the current line of the output file. The syntax of ADD operator is as follows:

```
ADD        species        [[-]field1[,field2...]]
```

Second field of the operator should contain a species or group name from those specified in the [species] section of the configuration file (5.4). Optional third field contains the comma-separated list of field names from those specified in the [fields] section of the configuration file (5.3); these fields of the gene table for given species will be copied to the output file in the order specified. If this field is omitted, *all* fields will be copied to the output file in the order they are stored in the table of genes. If the list follows a minus sign, all fields *except* the listed ones will be copied to the output file in the order they are stored in the table of genes. Field names may be specified with use of command line parameters in the form %*name* (chapter 3).

Let us define more exactly what species and gene are selected for copying the data from the table of genes:

- If the reference species *R* appears in ADD operator, the program selects currently tested gene *X* of the species *R*.
- If other species *A* is specified, for which any of IN operators was successfully verified earlier in the predicate, the program selects the gene that *satisfied* a condition of the IN operator, i.e., homolog *X'* in this species of the current gene *X* in the reference species. If no IN operator was satisfied for the reference species, a runtime error occurs.
- If other species *A* is specified, for which IN operator was executed and its condition was *not satisfied* (no homolog or witnesses exist in this species), the program will copy to the output file the equal number of empty fields.
- If the name of a species group appears in the second field of ADD operator, the program selects data for the species and gene that first satisfied the condition of previous IN operator or, otherwise, copies empty fields.

Above definitions are for the operation mode of lossgainRSL that requires one line of the output file per selected gene of the reference species. Such line contain data of the selected gene *X* and, possibly, its homologs *X'* in one or more other species (Fig. 3). The alternative mode is output of several additional lines for each witness such as *Y*, *Z* in the reference species *R* and their respective homologs in other species if any. If the latter mode is specified in the [mode] section of the configuration file (5.1), the ADD operator controls the output to all these line similarly. The order of lines in such synteny block is fixed: *X*, *Y*, *Z* (or *X*, *Y* or *X*). Thus, the sought-for gene of the reference species is always in the first line of the block.

Final recording of data prepared with the current gene *X* to the output file is performed only if the whole predicate is satisfied for the gene, i.e. YES label or the end of predicate is reached. Otherwise, the results of all ADD operators for the current gene are lost and no data is copied to the output file.

5.5.5 Verification of 2-species conditions and operators BOR/EOR/EOS, BAND/EAND

Let us describe the verification of 2-species conditions in different flavors by an example. Assume that a frog of the *Xenopus* genus is the reference species, and two more groups of species are considered, fishes and mammals, consisting of two species each. Ensembl data are assumed, so we do not use backward checking. We also choose 2 megabases as the neighborhood half-size in all species, and need to find each frog gene that is present in fish as an ortholog supported by two witnesses, but is absent from mammals even as a paralog of the ortholog and with at least one witness. In this example, the [species] section of the configuration file (5.4) can be as follows:

```
[species]
Xenopus
*Fish
Danio
Fugu
*Mammal
Mouse
Human
```

(A) The elementary variant of the predicate:

```
[predicate]
SET,2,2000K,7
IN          *Fish       -NO
ADD         Xenopus     -Protein
ADD         *Fish       Gene,Name
IN,1,,2     *Mammal     NO
```

Here, the first SET operator sets the standard values of parameters: two witnesses, neighborhood half-size of 2 megabases, orthologs to be considered as homologs. The next operator, IN, verifies for the current frog's gene *X* whether any fish has an ortholog *X′* of this gene, supported by two orthologous witnesses *Y′*, *Z′* in the specified neighborhood of *X′*. If neither fish has such *X′*, the current frog's gene *X* is skipped and the program proceeds to the next *X*; nothing is written to the output file. Otherwise, the ADD operator copies to the output file all but protein ID fields of the table of genes for current frog's gene *X*. One more ADD operator appends to the current line of the output file two fields with ID and name of *X′* gene found. (We assume that the program operates in X mode from 5.1, and field names are as per 5.3). Then the second IN operator checks if the gene *X* is present in mammals. Doing so, the operator modifies the standard parameter values: only one witness is sufficient and *X′* can be not only an ortholog of the gene *X*, but also a paralog of ortholog if any. If such gene and witness exists in either mammal, the program branches to the NO label (*X* is skipped, nothing is written to the output file, the program proceeds to the next *X*). Otherwise, the predicate end is reached, i.e. it is satisfied, and the program writes the composed line to the output file. The line contains all requested fields for the selected frog's gene and its homolog in a fish.

A disadvantage of the above predicate is that the output file does not clarify which fishes have the selected gene. Each line contains only one homolog of *X*, which was found earlier in a fish not saying about other fishes. The precedence of fishes is not fixed, but depends on order of records in the tables of orthologs and paralogs. Such limitation is even more obstructive, when groups of species under consideration include much more species than two ones like in our example.

(B) To improve the mentioned disadvantage, the more complex predicate with a fork can be used:

```
[predicate]
SET,2,2000K,7
HEADING     Xenopus,Contig,Start,End,Name,Description,Danio,Name,Fugu,Name
IN          Danio       -NotDanio
ADD         Xenopus     Gene,Contig,Start,End,Name,Description
ADD         Danio       Gene,Name
IN          Fugu
ADD         Fugu        Gene,Name
GOTO        Final
:NotDanio
IN          Fugu        -NO
ADD         Xenopus     Gene,Contig,Start,End,Name,Description
ADD         Danio       Gene,Name
ADD         Fugu        Gene,Name
:Final
IN,1,,2     *Mammal     NO
```

In this case, each field of the output line either contains or does not contain the data of the certain species, which allow assigning individual heading to each column of the file with use of HEADING operator (a variant of the SET operator). Recall that each operator must be entirely written in one line despite some lines may be lengthy such as this one.

Here, first IN operator reveals if the current gene *X* is present in zebrafish (*Danio rerio*), then the check forks: if the gene is not present in zebrafish, the program branches to `NotDanio` label. Otherwise, the following ADD operator copies necessary fields of the gene *X* of the reference species to output line in the order corresponding to above headings. Then one more ADD operator is executed, which adds two fields of *X'* ortholog in zebrafish to the same output line. The second IN operator checks whether the gene *X* is present in fugu. If so, two fields for its ortholog *X'* in fugu will be added to output line, otherwise ADD operator adds two empty fields. In both cases testing proceeds to `Final` label.

In another branch, starting from `NotDanio` label, the IN operator checks whether the gene *X* is present in fugu. If that is not the case, the program proceeds to NO label and nothing is written to the output file. Otherwise, first ADD operator copies necessary fields of the current frog's gene to output line. The second ADD operator adds two empty fields to that line because previous IN operator for zebrafish was unsatisfied. The last ADD operator adds two fields of *X'* ortholog in fugu to that output line. Thus, an output line with planned structure is composed in this branch too.

Starting from the `Final` label, the two branches merge again and the final check is performed like in the predicate (A). Depending on its result, the composed line is either written to output or lost if the gene *X* is rejected.

One can conclude that such method is rather inconvenient and results in excessive complication of the predicate, especially if the group consists of much more than two species. In addition, it is difficult to control how many species of the group must have the gene *X*. This is why the predicate definition language includes a pair of bracket operators, BOR/EOR (or BOR/EOS), that envelop several independent checks which success is determined by a quantitative threshold.

(C) The equivalent of the predicate (B) with use of these operators can be as follows:

```
[predicate]
```

```
SET,2,2000K,7
HEADING     Xenopus,Contig,Start,End,Name,Description,Danio,Name,Fugu,Name
BOR
IN          Danio
ADD         Xenopus     Gene,Contig,Start,End,Name,Description
ADD         Danio       Gene,Name
IN          Fugu
ADD         Fugu        Gene,Name
EOR,1       -NO
IN,1,,2     *Mammal     NO
```

The BOR operator has no arguments and should be specified in a line preceding the first alternative condition to be verified. In fact, the alternatives themselves are verified unconditionally: since IN operators lack labels, they are followed by next ADD operator(s) irrespective of the verification result. However, if the check was unsuccessful, added fields are empty. The aggregate condition is tested by the EOR operator, which follows the last alternative and has the following syntax:

```
EOR[,n]     [yes-label|–no-label]
```

Here *n* is a permissible minimum number of satisfied alternatives for the aggregate condition to be satisfied. If *n* is omitted, the default zero value means that the aggregate condition is always satisfied as well as if the label were not specified. Otherwise, the label is treated similar to the IN operator (5.5.2). The capability to change a single value of *n*, the more so as through a symbolic parameter directly in the command line, facilitates experimenting with various conditions for genes selection. There is also a variant of this operator with similar syntax, but slightly different semantics:

```
EOS[,n]     [yes-label|–no-label]
```

Here, the difference is that the aggregate condition is satisfied if *more than n* alternative tests are satisfied unlike the EOR operator, which uses a weak inequality. The pair of BOR/EOR (or BOR/EOS) operators can be used more than once in a predicate provided that such pairs do not nest.

Another pair of bracket operators, BAND/EAND, is somewhat dual. Unlike the EOR operator that counts the number of *satisfied* alternative conditions and compares it with a threshold, the EAND operator counts the number of *unsatisfied* alternatives:

```
EAND[,m]    [yes-label|–no-label]
```

where *m* is a permissible minimum number of unsatisfied alternatives for the aggregate condition to be *satisfied*. In other respects, this pair of operators is similar to the first one; however, the user should also keep in mind the difference: unsatisfied condition results in blank fields in the output line. Therefore, in the extreme case of BOR/EOR, when all alternatives are satisfied, the output line will contain *all* nonempty fields. On the contrary, in the extreme case of BAND/EAND, when all alternatives are unsatisfied, the output line will be *empty*. Nevertheless, this pair of operators can be more suitable for some predicates. The example in question unlikely illustrates this point, but let us provide without comment the same predicate using these operators:

(D) The equivalent of the predicate (C) with use of the BAND/EAND operators:

```
[predicate]
```

```
SET,2,2000K,7
HEADING    Xenopus,Contig,Start,End,Name,Description,Danio,Name,Fugu,Name
BAND
IN         Danio
ADD        Xenopus    Gene,Contig,Start,End,Name,Description
ADD        Danio      Gene,Name
IN         Fugu
ADD        Fugu       Gene,Name
EAND,2     NO
IN,1,,2    *Mammal    NO
```

5.5.6 Operators OVER/IN2/NEXT and verification of 3-species conditions

Besides the above described 2-species condition, the lossgainRSL program is capable to verify more complicated condition which involve two species, apart from the reference one, *R*. The first of these two species must have a homolog of the current gene *X* in the reference species supported by the required number of witnesses; such species *S* (and homolog) is referred to as *substitute*. The second species, *A*, either includes or lacks a homolog of the substitute gene (Fig. 5). This condition is referred to as *3-species* one; it is unsatisfied if either no substitute exist or another species does not have a homolog of any substitute.

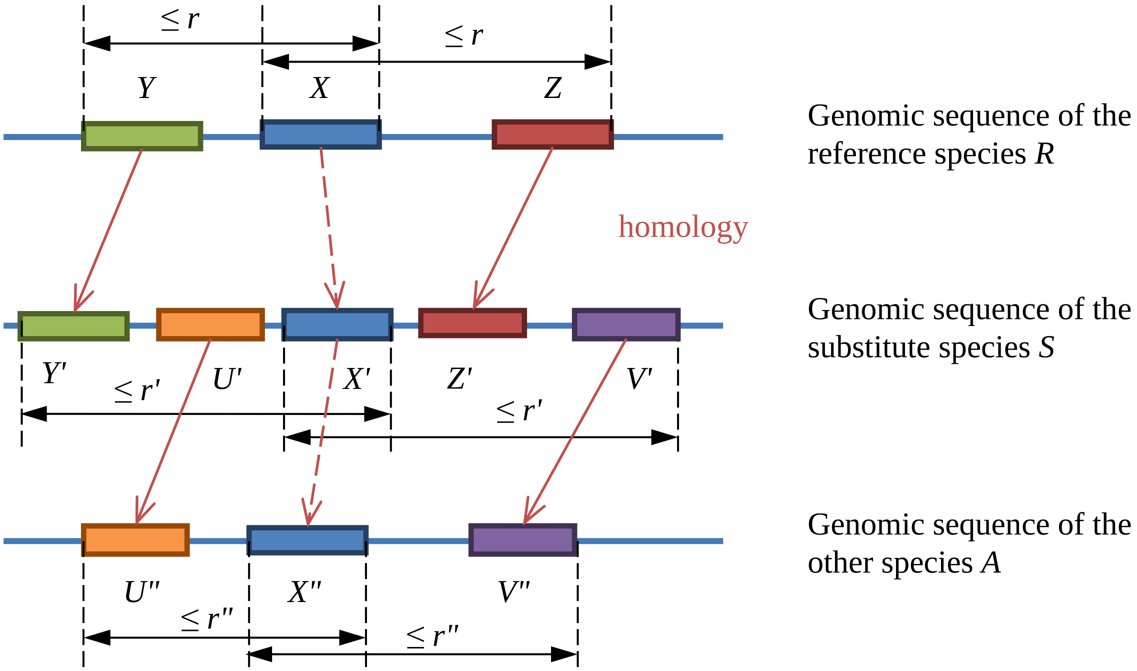
≤ r
≤ r
Y
X
Z
Genomic sequence of the reference species R
homology
Y'
U'
X'
Z'
V'
≤ r'
≤ r'
Genomic sequence of the substitute species S
U''
X''
V''
≤ r''
≤ r''
Genomic sequence of the other species A

Fig. 5. Satisfied 3-species condition for a certain variant of synteny.

If the homology relation in use makes up a transitive closure, the 3-species condition can be useful only in the situation, where the neighborhood of the *X* gene lacks homologs of the *U''* and *V''* genes, and the neighborhood of the *X''* gene lacks homologs of the *Y* и *Z*, otherwise a 2-species condition were satisfied. The existence of a substitute species with the *X'* gene whose neighborhood includes both pairs of homologs can be considered as an evidence of the conclusion that the *X* gene has the homolog, *X"*, in another species with the synteny conserved.

As in the 2-species condition, here the homology either means an orthology or also accounts for paralogy or relies on protein similarity etc.; and the certain number of witnesses and neighborhood size is specified for each species. These settings can be made in preceding SET operators (5.5.1) or directly in the conditional operators OVER and IN2 described below.

When verifying the 3-species condition, the HOLD1 parameter is normally used to control the permanence of *X*, *Y*, and *Z* genes during the evaluation of the *entire* predicate with current *X*. The usual value of 1 means that the same *X* shall be used in all tests within the predicate, whereas the witnesses *Y* and *Z* can be selected independently in each elementary condition.

The value of HOLD2 parameter allows the user to specify which genes are held in transition from the upper test to the lower one, see Fig. 5. This is a 3-bit value with the following meaning of individual bits:
1 – the same homolog of the *X* gene shall be used (designated as *X'* in Fig. 5),
2 – the same homolog of the first witness shall be used, i.e., *U'* = *Y'*,
4 – the same homolog of the second witness shall be used, i.e., *V'* = *Z'*.
As explained above, if the homology relation is transitive, which is true for OrthoDB orthogroups and protein clusters, but sometimes is broken in the Ensembl orthology and protein similarity matrices, the value of 1 should be specified for HOLD2.

Recall that the 2-species condition consists of the single IN operator which establishes the nested loop for selection of genes in the specified non-reference species *A* or species group (ref. to 5.5.2 for detail). In order to define the 3-species condition, three operators are used: OVER, IN2, and NEXT. The OVER and IN2 operators define two nested loops (outer and inner ones, respectively) for the selection of genes in the two non-reference species, and the NEXT operator points out the end of outer loop body. The order and syntax of these operators are as follows:

```
OVER[,w,r,b,k,s,e,l,h1,h2]      species1      [yes-label|–no-label]
IN2[,w,r,b,k,s,e,l,h1,h2]       species2      [yes-label|–no-label]
...
NEXT
```

The *outer loop* is defined by the OVER operator, where *species1* is the name of a substitute species *S* (or a group of such species). The substitute gene, *X'*, in this species has to be a homolog of the current gene *X* in the reference species *R*, and the neighborhood of *X'* has to include the witness genes, *Y'* and *Z'*, that are respective homologs of *Y* and *Z* genes in the neighborhood of *X*. The half-size *r'* of the neighborhood in the substitute species genome, number of witnesses, and homology variant, if differ from those initially set for the whole predicate, can be specified in preceding SET operator or just in the first field of the OVER operator.

The operator execution depends on the label specified. If the label is specified without preceding minus sign, the program jumps to this label, when first substitute gene is found irrespective of the inner loop conditions, otherwise the program proceeds to the next operator. Such variant of the condition seems to be senseless and is described here only for completeness. In typical situations, the label follows minus sign, i.e., *–no-label* is specified in the OVER operator. If this is the case, the program jumps to the label after all genes of the species (or species group) were tried, and none of them satisfies the conditions of both loops. If the label is omitted, it is interpreted as if a *–no-label* were specified pointing at the line following the NEXT operator.

The *inner loop* is defined by the IN2 operator, which is similar to the IN operator (5.5.2) with the only difference that instead of the *X* gene whose homologs to be selected in species *A* (or species group, argument *species2*), the current *X'* gene selected in the outer loop is used. The parameter values and labels in this operator are processed the same as in the IN operator. For instance, if a *yes-label* is specified (without preceding minus sign), and lossgainRSL finds a homolog of the *X'* gene along with witnesses in the *species2* genome, the program jumps to that label which can be inside or outside (such as NO) the outer loop. Otherwise, if *species2* lacks such homolog, *X"*, the program proceeds to subsequent operator of the outer loop body (behind ellipsis). In particular, it can be another IN2 operator that checks the current *X'* against other species, and so on.

The outer loop body ends with parameterless NEXT operator. It gives a signal for the interpreter to proceed back to the OVER operator with the next substitute gene *X'*.

The user should be careful with inclusion of side-effect operators (such as ADD) in the loop body, because of possible output of lines with unlimited length or at least variable format. The following rule of thumb is recommended to obey: once any data is written to the output file, the outer loop should be broken by jumping to a label outside the loop body.

Starting from version 6.31, the program lossgainRSL supports a backward check in the 3-species condition. The backward check is optionally carried out during the search of homologs independently in genomes *S* and *A*, that is achieved by variants of the above operators, OVERBK and IN2BK, respectively. Such backward check is performed similar to the INBK operator (5.5.2), and obeys the command line options `-b` and `--b` as well (see 3).

Note. Unlike the IN operator (5.5.2), operators OVER and IN2 in the current version of the lossgainRSL does not support the prefixes that enable the witness search in a semi-neighborhood of putative homolog. These features are to be implemented in future.

5.5.7 Predicate debugging

In fact, the predicate in the configuration file is a *program* which the lossgainRSL program interprets for each gene *X*. Like any program, the predicate can include errors leading to wrong results and failures in the lossgainRSL operation, e.g. crashes or looping. Some errors such as invalid syntax of operators, incorrect names or labels etc. will be found during the parsing of the configuration file, but other errors become evident only at runtime or even in the analysis of output results. Not attempting to suggest any universal method for debug, let us confine ourselves to describing the debug capabilities provided by the lossgainRSL and several typical ways.

During the program operation, it outputs to the console; the log can be redirected to a file by conventional means of the operating system. The program log is described in section 6.1. In case of any errors, the user should first verify in this log the following information:

- the command line used to run lossgainRSL e.g. options, parameters, and file names;
- the program version number;
- the program mode(s) as defined in 5.1;
- the list of species and species groups as well as the list of species in each group;

- the results of the predicate parsing. Ensure that after substituting the operator numbers for labels all jumps are correct as well as the species name and parameter values in each operator;
- the information on each data file read including its quantitative characteristics.

If the program crashes or gets in an endless loop, the console log informs about the last contig and gene of the reference species being processed, see the last line with the cKKK   xLLL format. Since gene *X* numbers are printed with the default step of 10, it is a good idea to repeat the run with the -g1 option (ref. to chapter 3).

The reported values of KKK and LLL can be used to request more detailed debug log. To this end, in the [mode] section of the configuration file the program debug mode N should be added in the first line, and one more line should be added containing KKK_LLL (see section 5.1 and Table 3 for detail). The debug log is directed to the standard stream *stderr* which is recommended to redirect to a file.

6 Output data

6.1 lossgainRSL operation log

The program log is mostly self-explanatory. Provided below is a sample of real log which is split into several blue-shaded fragments for explanation convenience.

```
Synteny and homology analysis utility (v6.31)

 arguments: for_manual.ini result.txt

Taxa: 6  Species: 41  Mode: H E X
```

In the first line, the program version number is shown. The next line shows arguments specified in the command line. Below is the number of species groups defined in the configuration file (6) and the total number of species (41) as well as active modes of the program (ref. to 5.1). When running in MPI environment, the number of parallel branches is also displayed.

```
*[1]: mus_musculus
*Human[1]: homo_sapiens
*Apes[5]: gorilla_gorilla nomascus_leucogenys pan_paniscus pan_troglodytes pongo_abelii
*Cercopithecidae[12]: cercocebus_atys chlorocebus_sabaeus colobus_angolensis_palliatus
macaca_fascicularis macaca_mulatta macaca_nemestrina mandrillus_leucophaeus papio_anubis
piliocolobus_tephrosceles rhinopithecus_bieti rhinopithecus_roxellana
theropithecus_gelada
*Other_mammals[13]: bos_taurus canis_lupus_familiaris delphinapterus_leucas
equus_caballus heterocephalus_glaber_female loxodonta_africana lynx_canadensis
monodelphis_domestica myotis_lucifugus panthera_leo physeter_catodon
rhinolophus_ferrumequinum ursus_maritimus
*Other_primates[9]: aotus_nancymaae callithrix_jacchus carlito_syrichta cebus_capucinus
microcebus_murinus otolemur_garnettii prolemur_simus propithecus_coquereli
saimiri_boliviensis_boliviensis
```

Then these groups are listed together the species they contain. Each group starts on a new line. Here, the groups are: the nameless * group that includes only mouse genome; the *Human group consisting of only human genome; the *Apes group consisting of five ape genomes (gorilla, gibbon, bonobo. chimpanzee, orangutan); the *Cercopithecidae group consisting of listed 12 Old World monkey genomes; the *Other_mammals and *Other_primates groups consisting of 13 and 9 genomes, respectively. This fragment is omitted if the command line includes `-q2` or `-q3` options.

Then the predicate parsing results are shown (also omitted if `-q2` or `-q3` options was specified).

```
The predicate (scenario) consists of 87 terms:
  0   SET                         1       2 2000000   7 10.000 0.001 1.0e-005 0.400   1   1
  1   SET                         2       2 5000000   7 10.000 0.001 1.0e-005 0.400   1   1
  2   IN          *Human          NO    3    1 5000000   7 10.000 0.001 1.0e-005 0.400   1   1
  3   SET                         4       2 2000000   7 10.000 0.001 1.0e-005 0.400   1   1
```

First line informs of the total number of operators recognized in the predicate definition (87). Thus, subsequent 97 lines of the log begin with the operator number. These numbers start from 0 and it is the SET operator (5.5.1). Each operator is reported in the log as several fields; some fields can be empty or have a special value depending on the specific operator. As for this SET operator, the line contains the operator number (0), its code, the number of the operator to execute next (1), and the parameter values: the required number of witness genes, WITNESS (2), the half-size of the neighborhood, RANGE (2 Mbp), ORTH (7), RANK (10), SCORE (0.001), EVAL (10^{-5}), LAMBDA (0.4), HOLD1 (1), and HOLD2 (1). These parameter values are used by default or can be temporarily changed by other SET operators such as one numbered 1. One could see from the log that the neighborhood half-size has been set to 5 megabases for a while. Then the IN operator (#1) checks whether the current mouse gene is present in human with one witness gene only for this operator. If yes, the program jumps to the NO label (the current gene is skipped). Otherwise, the program transits to the next operator, #3. That SET operator recovers the RANGE value of 2 megabases.

```
  4   ADD     mus_musculus    5    Fields: Gene ID,Region,Start,End,Strand,Label,Description
  5   BOR                       6
  6   IN      gorilla_gorilla    7       2 2000000   7 10.000 0.001 1.0e-005 0.400   1   1
  7   ADD     gorilla_gorilla    8       Fields: Gene ID
```

First ADD operator in the above fragment copies listed table fields of the current mouse gene to output line being formed. Shown here are the field names listed in the section [fields] of the configuration file (5.3), although their aliases could be requested for output. The next BOR operator informs that a series of checks starts from operator #6. At least a certain (yet unknown) number of these checks must be satisfied for the gene to be selected. The first check together with accompanying actions is included in this fragment of the log. The IN operator verifies whether the current mouse gene is present in the gorilla genome together with two witness genes. Independently of the verification result, the program proceeds to the operator #7, because there was no label in the IN operator. If a label were specified, two operator numbers stand here; one of them is always the next operator, and another number is somewhat else. As agreed, the program proceeds to the first number at satisfied check, and to the second one otherwise. If the operator does not fork like in this case, only one number of the next operator stands here. The effective parameter values follow in this line just the same as in the SET operator. They are equal to the initial ones. Second ADD operator (#7) in turn appends to the same line one more field (gene id) of the gorilla gene that first satisfies the check. If the check was unsuccessful (no gene identified), the empty field is appended. Of course, the user themselves chooses the field(s) to output.

```
  8   IN      nomascus_leucogenys   9       2 2000000   7 10.000 0.001 1.0e-005 0.400   1   1
  9   ADD     nomascus_leucogenys  10        Fields: Gene ID
 10   IN      pan_paniscus      11       2 2000000   7 10.000 0.001 1.0e-005 0.400   1   1
 11   ADD     pan_paniscus      12       Fields: Gene ID
```

```
12  IN     pan_troglodytes   13        2 2000000   7 10.000 0.001 1.0e-005 0.400  1  1
13  ADD    pan_troglodytes   14        Fields: Gene ID
14  IN     pongo_abelii      15        2 2000000   7 10.000 0.001 1.0e-005 0.400  1  1
15  ADD    pongo_abelii      16        Fields: Gene ID
16  EOR                      17  NO    4
```

The above fragment contains four more checks similar to the previous one but against different genomes. Each test is followed by the addition of the gene id from a genome under test, or empty field if no such gene is found. The last line in the above fragment contains the EOR operator, #16, (5.5.5) that ends this series of checks. The operator contains two labels, 17 and NO (the latter is virtual one which is not converted into number), and the parameter value *n*=4 which means that the predicate term is satisfied if at least 4 of those checks were successful (i.e., all but maybe one). In such case *this term* of the predicate is satisfied and the program proceeds to the next predicate term. Otherwise *the whole* predicate is unsatisfied and the program verifies the next mouse gene.

```
17  BOR                             18
18  IN         cercocebus_atys      19       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
19  ADD        cercocebus_atys      20       Fields: Gene ID
20  IN         chlorocebus_sabaeus  21        2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
21  ADD        chlorocebus_sabaeus  22        Fields: Gene ID
22  IN         colobus_angolensis_palliatus  23      2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
23  ADD        colobus_angolensis_palliatus  24       Fields: Gene ID
24  IN         macaca_fascicularis  25        2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
25  ADD        macaca_fascicularis  26        Fields: Gene ID
26  IN         macaca_mulatta    27       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
27  ADD        macaca_mulatta    28       Fields: Gene ID
28  IN         macaca_nemestrina  29       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
29  ADD        macaca_nemestrina  30       Fields: Gene ID
30  IN         mandrillus_leucophaeus  31       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
31  ADD        mandrillus_leucophaeus  32       Fields: Gene ID
32  IN         papio_anubis      33       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
33  ADD        papio_anubis      34       Fields: Gene ID
34  IN         piliocolobus_tephrosceles  35       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
35  ADD        piliocolobus_tephrosceles  36       Fields: Gene ID
36  IN         rhinopithecus_bieti  37        2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
37  ADD        rhinopithecus_bieti  38        Fields: Gene ID
38  IN         rhinopithecus_roxellana  39       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
39  ADD        rhinopithecus_roxellana  40       Fields: Gene ID
40  IN         theropithecus_gelada  41       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
41  ADD        theropithecus_gelada  42       Fields: Gene ID
42  EOR                             43  NO   2
```

This predicate term is quite similar, but it checks the gene against other set of 12 Old World monkey genomes and uses other parameter value of *n*=2.

```
43  IN         bos_taurus        44       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
44  ADD        bos_taurus        45       Fields: Gene ID
45  IN         canis_lupus_familiaris  46  2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
46  ADD        canis_lupus_familiaris  47       Fields: Gene ID
47  IN         delphinapterus_leucas  48   2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
48  ADD        delphinapterus_leucas  49       Fields: Gene ID
49  IN         equus_caballus    50       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
50  ADD        equus_caballus    51       Fields: Gene ID
51  IN         heterocephalus_glaber_female  52  2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
52  ADD        heterocephalus_glaber_female  53       Fields: Gene ID
53  IN         loxodonta_africana  54       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
54  ADD        loxodonta_africana  55       Fields: Gene ID
55  IN         lynx_canadensis   56       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
56  ADD        lynx_canadensis   57       Fields: Gene ID
57  IN         monodelphis_domestica  58       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
58  ADD        monodelphis_domestica  59       Fields: Gene ID
59  IN         myotis_lucifugus  60       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
60  ADD        myotis_lucifugus  61       Fields: Gene ID
61  IN         panthera_leo      62       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
62  ADD        panthera_leo      63       Fields: Gene ID
63  IN         physeter_catodon  64       2 2000000   7 10.000 0.001 1.0e-005 0.400   1  1
```

```
 64  ADD        physeter_catodon  65       Fields: Gene ID
 65  IN         rhinolophus_ferrumequinum  66       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 66  ADD        rhinolophus_ferrumequinum  67       Fields: Gene ID
 67  IN         ursus_maritimus   68       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 68  ADD        ursus_maritimus   69       Fields: Gene ID
 69  IN         aotus_nancymaae   70       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 70  ADD        aotus_nancymaae   71       Fields: Gene ID
 71  IN         callithrix_jacchus  72       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 72  ADD        callithrix_jacchus  73       Fields: Gene ID
 73  IN         carlito_syrichta  74       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 74  ADD        carlito_syrichta  75       Fields: Gene ID
 75  IN         cebus_capucinus   76       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 76  ADD        cebus_capucinus   77       Fields: Gene ID
 77  IN         microcebus_murinus  78       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 78  ADD        microcebus_murinus  79       Fields: Gene ID
 79  IN         otolemur_garnettii  80       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 80  ADD        otolemur_garnettii  81       Fields: Gene ID
 81  IN         prolemur_simus    82       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 82  ADD        prolemur_simus    83       Fields: Gene ID
 83  IN         propithecus_coquereli  84       2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 84  ADD        propithecus_coquereli  85       Fields: Gene ID
 85  IN         saimiri_boliviensis_boliviensis  86   2 2000000  7 10.000 0.001 1.0e-005 0.400  1  1
 86  ADD        saimiri_boliviensis_boliviensis YES       Fields: Gene ID
```

The above series of the predicate operators does not include any conditional branches and consequently does not affect the selection of mouse genes. It serves to output the information on the current mouse gene's presence or absence in other species under consideration. In particular, it allows the user to evaluate alternative conditions of mouse gene selection involving other species without recalculation. If the predicate verification reaches this fragment, the predicate is satisfied. Its last operator ADD is followed by transition to the YES label, and the output line(s) prepared is written to the output file. Otherwise, if the NO label was reached somewhere earlier, the predicate is unsatisfied and the line(s) is discarded. In both cases the program proceeds to the next gene *X* of the reference species starting with the operator #0.

In the following fragment the species are listed, whose genes will be shown in the output line, left to right. If the predicate generates output lines in a variable format, the variant with the greatest number of genes is displayed. This fragment is omitted if the command line includes `-q2` or `-q3` options.

```
Maximum 40 gene(s) per line: mus_musculus gorilla_gorilla nomascus_leucogenys
pan_paniscus pan_troglodytes pongo_abelii cercocebus_atys chlorocebus_sabaeus
colobus_angolensis_palliatus macaca_fascicularis macaca_mulatta macaca_nemestrina
mandrillus_leucophaeus papio_anubis piliocolobus_tephrosceles rhinopithecus_bieti
rhinopithecus_roxellana theropithecus_gelada bos_taurus canis_lupus_familiaris
delphinapterus_leucas equus_caballus heterocephalus_glaber_female loxodonta_africana
lynx_canadensis monodelphis_domestica myotis_lucifugus panthera_leo physeter_catodon
rhinolophus_ferrumequinum ursus_maritimus aotus_nancymaae callithrix_jacchus
carlito_syrichta cebus_capucinus microcebus_murinus otolemur_garnettii prolemur_simus
propithecus_coquereli saimiri_boliviensis_boliviensis
```

The following two lines provide the information on the table of genes that was read for the reference species unless the command line includes `-q3` option. The first line shows the input file name and path, and the second one displays its characteristics: total number of lines, number of unique genes, number of proteins (recall that lossgainRSL considers only protein-encoding genes), and the number of top level sequences (contigs for short) in the genome (35). Further c1 is the number of contigs that include only one gene (3), and c2 – that include exactly two genes (2).

```
Reading gene info file genes\mus_musculus.tsv ...
Lines read: 112608  genes: 22468  proteins: 67158  contigs: 35  c1: 3  c2: 2
```

As other tables of genes are read, similar data of other species are shown:

```
Reading gene info file genes\homo_sapiens.tsv ...
Lines read: 176163  genes: 24337  proteins: 117680  contigs: 1220  c1: 936  c2: 69
Reading gene info file genes\gorilla_gorilla.tsv ...
Lines read: 53486  genes: 21794  proteins: 45194  contigs: 327  c1: 271  c2: 25
Reading gene info file genes\nomascus_leucogenys.tsv ...
Lines read: 47561  genes: 20794  proteins: 40527  contigs: 336  c1: 225  c2: 49
Reading gene info file genes\pan_paniscus.tsv ...
Lines read: 52282  genes: 21210  proteins: 43232  contigs: 243  c1: 167  c2: 22
Reading gene info file genes\pan_troglodytes.tsv ...
Lines read: 60146  genes: 23534  proteins: 49949  contigs: 625  c1: 364  c2: 97
Reading gene info file genes\pongo_abelii.tsv ...
Lines read: 29435  genes: 20424  proteins: 21414  contigs: 53  c1: 0  c2: 2
Reading gene info file genes\cercocebus_atys.tsv ...
Lines read: 53602  genes: 20926  proteins: 46067  contigs: 549  c1: 129  c2: 26
Reading gene info file genes\chlorocebus_sabaeus.tsv ...
Lines read: 28077  genes: 19165  proteins: 19255  contigs: 119  c1: 65  c2: 9
Reading gene info file genes\colobus_angolensis_palliatus.tsv ...
Lines read: 47036  genes: 20617  proteins: 40399  contigs: 730  c1: 113  c2: 41
Reading gene info file genes\macaca_fascicularis.tsv ...
Lines read: 65967  genes: 22504  proteins: 49919  contigs: 147  c1: 58  c2: 27
Reading gene info file genes\macaca_mulatta.tsv ...
Lines read: 62443  genes: 21761  proteins: 48770  contigs: 175  c1: 108  c2: 27
Reading gene info file genes\macaca_nemestrina.tsv ...
Lines read: 54045  genes: 21060  proteins: 46238  contigs: 449  c1: 105  c2: 19
Reading gene info file genes\mandrillus_leucophaeus.tsv ...
Lines read: 47767  genes: 20841  proteins: 40903  contigs: 1538  c1: 313  c2: 153
Reading gene info file genes\papio_anubis.tsv ...
Lines read: 53012  genes: 21647  proteins: 45181  contigs: 418  c1: 275  c2: 70
Reading gene info file genes\piliocolobus_tephrosceles.tsv ...
Lines read: 54302  genes: 24588  proteins: 41245  contigs: 3537  c1: 2848  c2: 190
Reading gene info file genes\rhinopithecus_bieti.tsv ...
Lines read: 52671  genes: 20966  proteins: 43730  contigs: 2510  c1: 1007  c2: 248
Reading gene info file genes\rhinopithecus_roxellana.tsv ...
Lines read: 53493  genes: 21289  proteins: 45897  contigs: 2777  c1: 1001  c2: 343
Reading gene info file genes\theropithecus_gelada.tsv ...
Lines read: 43834  genes: 22515  proteins: 36976  contigs: 414  c1: 337  c2: 39
Reading gene info file genes\bos_taurus.tsv ...
Lines read: 43267  genes: 21880  proteins: 37538  contigs: 128  c1: 66  c2: 13
Reading gene info file genes\canis_lupus_familiaris.tsv ...
Lines read: 55790  genes: 20257  proteins: 45094  contigs: 221  c1: 134  c2: 33
Reading gene info file genes\delphinapterus_leucas.tsv ...
Lines read: 33898  genes: 19091  proteins: 30992  contigs: 262  c1: 124  c2: 11
Reading gene info file genes\equus_caballus.tsv ...
Lines read: 54352  genes: 20955  proteins: 44934  contigs: 299  c1: 178  c2: 48
Reading gene info file genes\heterocephalus_glaber_female.tsv ...
Lines read: 40018  genes: 20774  proteins: 28984  contigs: 366  c1: 92  c2: 17
Reading gene info file genes\loxodonta_africana.tsv ...
Lines read: 28849  genes: 20033  proteins: 25635  contigs: 583  c1: 228  c2: 81
Reading gene info file genes\lynx_canadensis.tsv ...
Lines read: 38883  genes: 19131  proteins: 35180  contigs: 36  c1: 5  c2: 5
Reading gene info file genes\monodelphis_domestica.tsv ...
Lines read: 50160  genes: 21384  proteins: 36557  contigs: 425  c1: 258  c2: 59
Reading gene info file genes\myotis_lucifugus.tsv ...
Lines read: 26842  genes: 19728  proteins: 20719  contigs: 2418  c1: 1169  c2: 305
Reading gene info file genes\panthera_leo.tsv ...
Lines read: 34373  genes: 19550  proteins: 31178  contigs: 459  c1: 262  c2: 58
Reading gene info file genes\physeter_catodon.tsv ...
Lines read: 35311  genes: 19717  proteins: 32206  contigs: 1279  c1: 676  c2: 243
Reading gene info file genes\rhinolophus_ferrumequinum.tsv ...
Lines read: 36358  genes: 19533  proteins: 33717  contigs: 60  c1: 13  c2: 6
Reading gene info file genes\ursus_maritimus.tsv ...
```

```
Lines read: 39675  genes: 18724  proteins: 34052  contigs: 463  c1: 124  c2: 24
Reading gene info file genes\aotus_nancymaae.tsv ...
Lines read: 51062  genes: 20412  proteins: 42510  contigs: 898  c1: 268  c2: 66
Reading gene info file genes\callithrix_jacchus.tsv ...
Lines read: 61545  genes: 22615  proteins: 43000  contigs: 545  c1: 483  c2: 13
Reading gene info file genes\carlito_syrichta.tsv ...
Lines read: 38316  genes: 18398  proteins: 31791  contigs: 5933  c1: 2658  c2: 1086
Reading gene info file genes\cebus_capucinus.tsv ...
Lines read: 48350  genes: 20317  proteins: 40677  contigs: 1037  c1: 199  c2: 86
Reading gene info file genes\microcebus_murinus.tsv ...
Lines read: 45983  genes: 18895  proteins: 38078  contigs: 76  c1: 32  c2: 2
Reading gene info file genes\otolemur_garnettii.tsv ...
Lines read: 28567  genes: 19506  proteins: 19986  contigs: 525  c1: 151  c2: 38
Reading gene info file genes\prolemur_simus.tsv ...
Lines read: 43803  genes: 20354  proteins: 38056  contigs: 2231  c1: 888  c2: 205
Reading gene info file genes\propithecus_coquereli.tsv ...
Lines read: 37914  genes: 17925  proteins: 32202  contigs: 810  c1: 185  c2: 52
Reading gene info file genes\saimiri_boliviensis_boliviensis.tsv ...
Lines read: 48819  genes: 19380  proteins: 40695  contigs: 320  c1: 45  c2: 16
```

This information is omitted if the command line includes `−q3` option (but is still preserved with `-q0`…`-q2` options). The same apply to subsequent lines which inform of reading the table of orthologs for the reference species and the tables of paralogs if needed. In this example, the only table of mouse orthologs is needed. In the end of each line, the total number of ordered pairs of orthologous (or paralogous) genes is shown. This display is for the selected mode H (input data from Ensembl). In other program modes, different input files are read.

```
Reading orthologs from orthologs\mus_musculus.tsv ...   747042
```

Then the two following lines are displayed. The first one reminds the reference species and the number of top level sequences ("contigs" for short) in its genome (35) as well as the number of genes (22468). The second line shows how many contigs and genes will be actually tried in this task. Since two witness are required, contigs with only one or two genes are skipped.

```
mus_musculus in total: 35 contigs, 22468 genes.
With several genes: 32 contigs, 22465 genes.
```

After that, the search of genes is initiated, during which the lossgainRSL program outputs to console the messages like this:

```
c5   x130
```

where first number (5) is the serial number of contig under test in the reference species, and second number (130) is the serial number of *X* gene in this contig. These numbers start from 0 and depend on the table of genes. The gene numbers are displayed with a step specified in the command line by `−g` option. This information can be helpful for debugging (ref. to 5.5.7). After completion, the number of identified genes and timing are provided:

```
mus_musculus candidate genes selected: 230

54s: Completed OK.
```

6.2 Output file

The results of the lossgainRSL operation are written to the output file with a name and path specified in the command line (see chapter 3) or set by default – result.txt in the working directory. The existing file will be overwritten.

The file is written in the tab-separated value (TSV) format; the first line contains the field (column) names. It is assumed that the number and meaning of the fields are the same for all output lines, although the user can specify a predicate that writes the lines in a variable format. The priority names are those specified by the HEAD (or HEADING) keyword in the predicate (5.5.1), in which case the user is responsible for proper correspondence of the names and columns. If the names are not specified in the predicate, the aliases (if any) or field names from the [fields] section (5.3) of the configuration file are used.

If the program mode X (5.1) was specified, each identified gene of the reference species is presented by one output line consisting of the fields added by a series of ADD operators during the verification of that gene. If the X mode was not specified, the program additionally writes one line per witness that contains similar data for witnesses and their homologs in other species. Moreover, an empty line is written after each synteny block if I is present in the program mode. In this case two more fields can be added, by specifying the mode J, in the end of each line to facilitate sorting of the synteny blocks as a whole (see 5.1 for detail). In addition, if the lossgainRSL operates in MPI environment, each output line is automatically appended with a field named 'rank' where parallel branch number is provided for debug purpose.

The output file samples are provided in the test, http://lab6.iitp.ru/en/lossgainrsl/example.zip. The output file allows for import to the Excel spreadsheet for convenient post-processing and analysis. An example of the output file imported to the Excel is provided in the 'Result' sheet of the file http://lab6.iitp.ru/ru/lossgainrsl/dataformat.xlsx.

7 Using other input data sources

7.1 Addition of genomes not included in the Ensembl

The Ensembl database already includes the complete genomes of above 300 species, and further increases. Nevertheless a task can appear, where *few* (implying one-two) species have to be added that are not included in the Ensembl yet. In principle, it is possible, but involves a laborious work to draw up new tables and augment existing ones, manually or by ad hoc scripts or programs.

Below we outline a work for addition of a *new* genome to existing input data (see chapter 4). In order to add several new genomes, this should be done by turns for each of them. Let us accentuate that a new reference species cannot be added in this way and should be taken from the Ensembl, at least in the current program version. If the Ensembl database lacks the reference species, *all* information on gene orthology should be obtained from other sources (7.2).

GenBank is virtually the most important source of new complete genomes. In general, it contains all necessary data, but sometimes in insufficiently unified and disembodied form, which makes the drawing up tables 4.1–4.3 a challenge. The genome in the GBFF flat file format seems to be most convenient for manual work described below; the lossgainRSL user also can develop himself a program or script to use some other source.

7.1.1 Manual preparation of a new table of genes

First of all, the user should prepare the table of genes for the new species. Such table must have the format described in 4.1 including the field headings. It is required that all tables of genes use consistent field names. Manual preparation of the table could be easier to make in Excel using the existing table of genes as a template. Let us point out the data sources for the fields listed in Table 1.

Chromosome/scaffold name

In the LOCUS line, which initiates each section of the genome in GBFF format, the accession code of the sequence is provided. This code does not include a version number, and can be safely used as a name of the top level genomic sequence. However, for the results to be more informative we recommend to examine also the next line (DEFINITION). For example, if it reads “Drosophila melanogaster chromosome 2R”, more suitable name would be 2R. The name assigned to the sequence should be used in the table entries for all proteins/genes annotated in this section of the GBFF-formatted file. The section lines from FEATURES to ORIGIN are subject to further analysis.

Gene stable ID, Gene name

Only protein-coding genes should be inserted in the table. Such genes are represented by entries gene, mRNA, and CDS in the file. The gene id appears with the /gene tag in each of these entries; we recommend to use it for the Gene name field, and assign the unique gene identifier by any formal method like Gene IDs in Ensembl.

When entering a gene in the table, its size and structure should be considered. If the gene consists of several exons separated by long introns or spacers, sometimes it may be better split into several separate “genes” with individual names. If the gene is accompanied with multiple amino acid sequences in several CDS entries according to the splicing variants, they should be entered to the table as separate rows with same Gene ID but different Protein IDs.

Gene start (bp), Gene end (bp), Strand

The gene start and end positions in the nucleotide sequence are provided in gene, mRNA, and CDS entries. If the coordinates follow the keyword “complement”, the Strand field should contain -1, otherwise 1.

Gene description

For this column, one can use data fields in the gene, mRNA, and CDS entries under the /note and /product tags. The formal procedure unlikely can be proposed, which usually presents no difficulty at manual compiling this column.

Protein stable ID

The protein identifier should be assigned to each amino acid sequence that is represented in the CDS entry under the /translation tag; a formal procedure should be established for this purpose. As mentioned above, the table of genes for the new species can include multiple lines with the same Gene ID, but different Protein IDs corresponding to the alternative splicing variants.

When drawing up the table of genes, it is very important to collect the file of proteins for the new species. The file should be prepared in the FASTA format, where the sequence is taken from the /translation tag, and the name is a previously assigned protein id. The user also can download the existing FASTA-formatted file with all protein sequences of the new species, and then substitute the assigned protein names for the sequence headings. Or, vice versa, use the first field of the heading as the protein id in the table of genes.

The recommendations of this section are preliminary ones to be refined from the future experience of new species addition.

7.1.2 Importing the table of genes from RefSeq

The procedure described in this section is applicable to the cases where the complete genome of the new species is present among the NCBI reference sequence (RefSeq), which takes place for many GenBank species, see ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/. A reference assembly usually includes the file "feature_table", from which the required table of genes can be easily obtained. For example, the file for Pacific walrus *Odobenus rosmarus divergence* is available at URL ftp://ftp.ncbi.nlm.nih.gov/genomes/all/annotation_releases/9708/101/GCF_000321225.1_Oros_1.0/GCF_000321225.1_Oros_1.0_feature_table.txt.gz, and the table of genes in accordance with 4.1 can be obtained by the python script MakeGeneTable.py authored by Oleg Zverkov. The script is included in `gbinput` package downloadable from the lossgainRSL program webpage, http://lab6.iitp.ru/en/lossgainrsl/. The script should be started from a folder where abovementioned feature table has been downloaded; the gene table will be formed in the same folder. For different folders and species lines 4, 5 of the script should be edited respectively.

7.1.3 Augmenting the ortholog and paralog tables by *addspecies* utility

If the predicate (5.5) does not include 3-species conditions, the table of orthologs (4.2) is required only for the reference species, otherwise also for each putative substitute species. If a new non-Ensemble species is added, this table must be augmented by the records for all pairs of orthologous genes from the reference and new species. In addition, if the predicate accounts for paralogous genes, the table of paralogs (4.3) may be required for the new species. To this end, the ancillary utility **addspecies** is available within `gbinput` package at the lossgainRSL main page (http://lab6.iitp.ru/en/lossgainrsl/).

As a prerequisite to the utility, the user should apply BLASTP to obtain the scores for the local alignment of all pairs of proteins from the reference and new species as well as from the reference and new species apart. The ordered pairs (i.e. with transpositions) should be considered, because in general case the score can depend on which of the two species acts as a target. Prior to using BLASTP, all protein sequences for both species in FASTA format must be obtained. For Ensembl genomes, these files are available at their ftp server in the pep subdirectory for each species, for example, at ftp://ftp.ensembl.org/pub/current_fasta/mus_musculus/pep/ for mouse. For RefSeq and GenBank genomes, similar files are available in the assembly-related directory e.g. GCF_000321225.1_Oros_1.0_protein.faa.gz for walrus (complete path to the directory see in 7.1.2). Using of BLASTP is not discussed here; the expected results are the four protein similarity matrices similar to those described in section 4.4. These matrices should be named *spec1-spec2*.*, *spec2-spec1*.*, *spec1-spec1*.*, and *spec2-spec2*.*, where *spec1,2* are the names of the reference and new species listed in the [species] section of the configuration file (0). Let us stress that protein identifiers in these protein similarity matrices must be *exactly* the same as those used in the table of genes for the reference and new species. It is essential as existing FASTA files can contain the protein names including a version number, while IDs in the table of genes normally lack version numbers.

The program "addspecies" was written in C++ and is a command line utility for Windows/linux (32/64 bit). The utility uses the following command line syntax:

`addspecies [`options`]` *source target* `[`*config*`]`

The utility name in Windows 64-bit environment is `addspecies64`. The required arguments are as follows: *source* is the name of a new species to be added, *target* is the name of an existing species (the reference or substitute one). All species names must be listed in the [species] section of the configuration file that can be one used for the lossgainRSL program (chapter 5) with the addition of [add] section before or after any other section. The [add] section should be as follows (examples of the numerical values are shown):

```
[add]
EVALUE        1e-10
ORTHOLOG      0.35
PARALOG       0.35
```

The name of the configuration file including optional path can be specified by the *config* argument in the command line; default is `config.ini` in working directory. Note that lossgainRSL ignores the [add] section, and "addspecies" ignores unnecessary sections from those described in chapter 5. Thus, in practice both programs can share the same configuration file.

The parameters in the [add] section have the following meaning:
EVALUE – the E-value cut-off for the alignment of two proteins; the pairs with greater E-value are skipped.
ORTHOLOG – the minimum acceptable weight for the similarity of two proteins from *source* and *target* species, calculated by the formula $w = 2S_{st} / (S_{ss} + S_{tt})$, where S_{st} is the raw score for the alignment of the two proteins (one from *source*, another from *target*), and S_{ss}, S_{tt} are the raw scores for the alignment of each protein with itself. If the weight exceeds the threshold specified, respective genes are considered as orthologs.
PARALOG – the minimum acceptable weight for the similarity of two proteins from the new (*source*) species to consider them as paralogs. The weight is calculated by the above formula, but for the two proteins from the same species.

The *addspecies* utility supports the following case-insensitive options:

-E*n* Sets the value of *n* as E-value cut-off. If this option is specified, the parameter EVALUE in the [add] section of the configuration file is ignored. Default cut-off value equals 1E-10 (i.e. 10^{-10}).

-G*n* The step for displaying gene numbers in each contig. Default is 10, zero value cancels displaying of gene numbers.

-M If this option is specified, the program computes and stores the four matrices of E-values and raw scores between the genes of *source* and *target* species. These matrices are similar to above protein similarity matrices, except for using gene IDs instead of protein IDs. Each entry contains the minimum E-value and maximum raw score over all pairs of proteins encoded by given genes. Each matrix is sorted in the ascending order of first gene id and in the descending order of raw score. The directory to write the matrices should be specified in the [data] section of the configuration file (5.2) with a special data type M (in addition to those listed in Table 4) e.g. as the line:

```
M        matrix\*-*.tsv
```

In such case, the matrix files will be named as *species1-species2.tsv*, where *source* and *target* will be substituted in all four combinations for *species1* and *species2*.

-U The same as -M option, but the matrices are not ordered.

-Q*n* This option is intended for the program termination at intermediate points. If the value of 0 is specified, the *addspecies* utility writes the gene similarity matrices (if specified -M or -U) and finishes. If the value of 1 is specified, in addition the program augments the table of orthologs for the *target* species. If the value of 2 or greater is specified, in addition the program creates the table of paralogs for the *source* species (default mode).

-O*n* Sets a new threshold for the weight of orthologs. If this option is specified, the parameter ORTHOLOG in the [add] section of the configuration file is ignored. By default, the threshold of 0.5 is used.

-P*n* Sets a new threshold for the weight of paralogs. If this option is specified, the parameter PARALOG in the [add] section of the configuration file is ignored. By default, the threshold of 0.5 is used.

-W*file* If this option is specified, the program writes identified pairs of orthologous genes from the *source* and *target* species to the separate file with the name specified. By default, the new orthologous pairs are appended to the existing table of orthologs for the *target* species. In both cases it uses the directory specified for O data type in the [data] section of the configuration file (5.2).

As for the table of paralogs for the new (*source*) species, the program uses the directory specified for P data type in the [data] section of the configuration file (5.2), and the file is named as other tables of paralogs.

In order to use the *addspecies* utility, first the user should make necessary modification of the configuration file (in particular, specify the new species name). Then, the four protein similarity matrices (4.4) should be obtained using BLASTP. These matrices should be properly named and placed in the directory specified for A data type in the [data] section of the configuration file (5.2). The optimum cut-off values for the weights and E-value are selected empirically. The algorithm that has been implemented in the program is rather experimental and preliminary one.

7.2 Using alternative information on gene homology

The best method for the ortholog inference is unknown; different methods yield different results. The information on orthologs and paralogs provided in the Ensembl database also gives rise to complaints in some cases. The lossgainRSL program allows the user to take advantage of other orthology inference methods, but this should be always done *as a whole*, i.e. for all species and genes under study. For example, one may use the protein similarity matrices (4.4) or the table of clusters (4.5) instead of the ortholog/paralog tables from the Ensembl, still using the tables of genes (4.1) from the Ensembl.

The user can decide against using Ensembl entirely and input from the table of orthologous groups (4.5) composed from the OrthoDB data. Or the user can compile the gene tables (4.1) and protein similarity matrices (4.4) wholly from NCBI data as described in.

Generally, when running the program with different sources of orthology, different lists of genes will be obtained which provides the users with new opportunities. Let us discuss two of them. If the minimum overprediction is a priority, and the user aims at as short as possible lists of more reliable genes having the desired properties e.g. for wet experiment, it can be achieved by intersecting of the lists obtained for different orthology inference methods. And vice versa, one can decrease the underprediction by joining independent lists of genes.

Besides such rigid set-theoretic operations as the intersection and union, one can apply more slack procedures such as voting if three or more sources of orthology are used. New study is ahead to integrate existing and future conceptions of gene orthology, especially in distant species.

7.3 Using input data from only NCBI database

Gene orthology data available in NCBI database do not pretend to be complete and often just redirect to the OrthoDB for information on orthologs. Therefore, the most suitable input data for the lossgainRSL program are the gene tables (4.1) and protein similarity matrices (4.4). To identify true homologs more reliably, the matrix should include, for each gene *X* of species *R*, as many its best homologs *X'* in *every* species *A* as possible. For instance, up to 10-15 homologs that are good enough, e.g. satisfy the E-value cut-off of 0.001 or less. This enables future choosing a proper value of the parameter *k*, greatest rank of a hit (Table 5) and also using of the backward check to identify true homologs (5.5.2). Below is a typical action sequence for the preparation of input data.

Create folders and files

Create working directory (e.g. `work)` and copy a needed version of lossgainRSL to that directory as well as ancillary utilities and scripts from `gbinput` package downloadable from the program main page, http://lab6.iitp.ru/en/lossgainrsl/. In this directory, create the future configuration file, `config.ini` (see chapter 5), and one more text file `species.txt` described below. The first line of the file must contain the field heads, “filename”, “protein” and “feature”, separated by tab character. Then create several subdirectories of the `work` directory by the following commands:

```
    mkdir blastdb
    mkdir dup
    mkdir feature
    mkdir fasta
    mkdir gene
    mkdir hits
    mkdir homolog
    mkdir protein
```

Choose species and get data files from NCBI

The set of species depends on the study, but well-assembled genomes from RefSeq are favored. First, an unique id not containing blanks has to be assigned to each species, such as scientific binomial name e.g. `pan_troglodytes`. Then the accession of the chosen genome should be mentioned, GCF_002880755.1 in the case, which allows accessing to respective directory of the NCBI FTP site:

ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/002/880/755/

The directory usually contains only a symbolic link the genome data folder, where the two gzipped files are to be downloaded from. These files are: (1) amino acid sequences of all known proteins, i.e., the proteome, and (2) the genome feature table, which in this case are named

GCF_002880755.1_Clint_PTRv2_protein.faa.gz
GCF_002880755.1_Clint_PTRv2_feature_table.txt.gz

For each species, the first file should be unpacked to the folder `protein`, and the second file to the folder `feature`. The current line of the `species.txt` file should contain three tab separated fields: the species id, the name of the unpacked proteome file, and the name of the unpacked feature table. Example of the `species.txt` file imported to the Excel is provided in the 'Species_list' spreadsheet of the file http://lab6.iitp.ru/ru/lossgainrsl/dataformat.xlsx. Apart from this, the current species id should be added to the `[species]` section of the `config.ini` file (5.4).

Build the tables of genes

From the `work` directory, run python script `gb2en.py`, that is similar the script from 7.1.2 enhanced for serial processing of the feature tables of *all* genomes listed in `species.txt` rather than one specified genome. For this, a python interpreter version 3.10 or above must be installed in the computer. As a result of running the script, the directory `gene` will contain TSV formatted files *species*`.tsv`, where *species* is a species id. Each file conforms to the specification in 4.1.

Preprocess the proteome files

The preprocessing is aimed at indexing of proteins and elimination of copies of the same amino acid sequence with different name. From the `work` directory, run the command:

```
fastaprep -r 0 -u 0 -w 1 species.txt species.txt
```

As a result, the directory `fasta` will contain files in FASTA format named *species*`.fas` and directory `dup` – working tables named *species*`.tsv`, where *species* is a species id. Each FASTA file is normally smaller than corresponding proteome in folder `protein`, because it lacks some proteins and unused data in sequence headers.

Find protein hits

There are two variants of possible actions:

Option 1

Using 'makeblastdb' program from the package BLAST v2.12.0+ or above, build a standalone database for each species. For example, the database for the above species is built by the command:

```
makeblastdb -in fasta/pan_troglodytes.fas -dbtype prot -out blastdb/pan_troglodytes
```

For other species use similar commands with the `-in` и `-out` arguments changed respectively. Then for each pair of species, *species1* and *species2*, without respect of order and including the case of equal identifiers, run the command:

```
blastp -outfmt "6 qseqid sseqid score evalue" -evalue 0.1 -max_target_seqs 1000
-max_hsps 1 -use_sw_tback -query fasta/species1.fas -db blastdb/species2 -out
hits/species1-species2.tsv
```

This option is practical if less than 25-30 species are considered so up to several hundred commands are to be composed and run.

Notes**:**

1. Execution of the above command can take dozens of hours, so a supercomputer is recommended for parallel running of many such commands for different pairs of species. If a multicore CPU is used, the execution can be speed up by addition an argument `num_threads` *n* to the command, where *n* is integer number greater than 1.
2. If more than 10-12 species are considered, it is recommended to split the folder `hits` into subdirectories named by each species id and specify the last argument of the command line in the form `hits/`*species1*`/`*species1*–*species2*`.tsv`, for the filesystem not to slow down due to great number of files in single directory `hits`.

3. As a faster alternative for BLAST+, the package DIAMOND may be applied. It should be run with the parameters that provide in the `hits` directory the files of the same format and contents as blastp provides.

Option 2

Merge all FASTA formatted files using the OS command:

```
copy /a fasta/*.faa /b fasta/common.fasta   (Windows)
cat fasta/*.faa >fasta/common.fasta          (linux)
```

Build common database for all species under consideration:

```
makeblastdb -in fasta/common.fasta -dbtype prot -out blastdb/common
```

Then for each species *species* in turn run the command:

```
blastp -outfmt "6 qseqid sseqid score evalue" -evalue 0.1 -max_target_seqs 10000
-max_hsps 1 -use_sw_tback -query fasta/species.fas -db blastdb/common -out
hits/species.tsv
```

This option better suits for numerous species, when number of their pairs becomes too big. Here, the number of commands to run equals the number of species, but each command will take more time than with the first option, proportionally to the total number of species. In addition, if more than 30-50 species are considered, it is recommended to increase the value of `-max_target_seqs` parameter up to e.g. 30000-50000.

Irrespective of option chosen, the finding of protein hits with `blastp` is recommended to perform on a supercomputer. If the `diamond` program is used, a powerful PC or a server will do.

Build gene similarity matrices

These matrices are the special case of the protein similarity matrices (4.4) for faster processing. One can build them with use of the `posthit` command line utility for 64-bit Windows or linux written in C++ with support of MPI-based multiprocessing if available. In single CPU mode run the command:

```
posthit -b 1 -c 0 -i hits/*.tsv -n 10 -r 2 -w 0 list.txt
```

As a result, the matrix for each species will be built in the directory `homolog`. Each entry of the matrix contains all six fields listed in Table 2 and field #3 is the raw score value. To get the matrix of weights in this field, one should specify `-w 1` in the above command.