SCJ suite — User Manual

P. Biller P. Feijão J. Meidanis

January 2, 2013

Abstract

We have developed a variety of algorithms for solving rearrangement genome problems under the Single-Cut-or-Join (SCJ) model [5]. We named this project *SCJ suite*, and the code is written in Java. The project includes algorithms for simulating rearrangement evolution, for parsing Newick files, and Perl scripts for assisting in the analysis of results.

The purpose of this document is to explain the basics of how to use *SCJ suite*, and to give a general overview of the project structure, composed by several modules. We gave special attention to one of these modules, called *compareTrees*, used in the experiments detailed in the first author's M.Sc. thesis [2] (available in Portuguese only) and also in the experiments of a related paper [3].

Contents

1 Getting SCJ suite

The jar file and source code are available in:

`http://www.ic.unicamp.br/~meidanis/PUB/Mestrado/2010-Biller.`

Download the files and save the extracted files preferably in a directory that does not have spaces in its full pathname.

The files that compose the project, compressed in the file "SCJ-suite.zip", are organized as follows:

- *scj.jar:* executable file;
- *data:* this directory contains all the data used in the experiment detailed in Priscila Biller's M.Sc. thesis [2], available in Portuguese only. The subdirectory "simulatedTrees" has different tree topologies, used in simulations of rearrangement evolution. These trees were generated according to the beta-splitting model [1]. Each file corresponds to a tree described in the Newick format. The subdirectory "input" has sets of genomes, where each file corresponds to a real dataset, such as Campanulaceae [4] or Protostomes [6].
- *graficos:* directory where the results are saved;
- *scriptsPerl:* directory that contains auxiliary scripts used in the analysis of results.

2 Prerequisites

The software was tested on Windows and on Linux operating systems. To run it, you need the following additional tools:

- *Perl*, to execute auxiliary scripts;
- *Java*, to execute the main project. *SCJ suite* requires Java 5 or a higher version.

3 Input files

3.1 Genomes represented as permutations

SCJ suite works either on random genomes generated internally, or on genomes given as input by the user. If you need to use a specific set of genomes, it is possible to create a text file containing your dataset, where each genome is seen as a permutation of its genes. Consider a set of m unichromosomal genomes, linear or circular, with n genes. The genes are represented by the numbers $1, 2, \ldots, n$ or $name_1, name_2, \ldots, name_n$. A gene i and its reverse complement are represented by $+i$ and $-i$, respectively. A genome is represented as a permutation of genes or their reverse complement.

Keep in mind that *SCJ suite* can create random multichromosomal genomes, but does not accept this kind of genome in the input file. The project also does not deal with datasets containing variable gene content.

Here is how to create a text file with a genome dataset:

- The first line of the file must have the format "m n [s]", where m is the number of genomes, n is the number of genes, and $[s]$ indicates that the genes are represented by their names instead of numbers. The first two arguments are mandatory and the third argument is optional. Arguments are separated by spaces. The absence of the third argument indicates that genes are represented by $1, 2, \ldots, n$, and its presence indicates that genes are represented by $name_1, name_2, \ldots, name_n$;
- If argument $[s]$ is present in the first line, the second line must specify the gene names: "$name_1$ $name_2$ ... $name_n$". The gene names are composed by letters and numbers, and are separated by spaces;
- The remaining lines contain the genomes. Each genome is represented in three lines:
  - The first line must contain the genome name, following the format "$>genome\_name$", as in FASTA format, where the description of a genome begins with its name. The genome name is composed by letters, numbers, hyphens or underscores, and is recognized only when the symbol ">" precedes it.
  - The second line must specify whether the genome is "linear" or "circular";
  - The third line must contain the order and orientation of genes in the genome. Genes are separated by spaces, and each gene is represented by "*orientation label*", where the argument "*orientation*" can be "+" or "−", and the argument *label* can be a number or a name (if the option $[s]$ is indicated in the first line) that identifies the gene. If you use gene names, valid names are those specified in the second line of the file, case-sensitive.

An example of a valid dataset is:

```
3 20
>rco
linear
+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +13 -12 +14 +15 +16 +17 +18 +19 +20
>raf
linear
+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 -13 +14 +15 +16 +17 +18 +19 +20
>rma
linear
+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11 +12 +13 +14 +15 +16 +17 +18 -19 +20
```

In this example, the first line shows us that there are 3 genomes (later identified as *rco*, *raf* and *rma*), each one with 20 genes. The omission of the argument [*s*] indicates that the genes are represented by numbers. In this case, all genomes are linear. Another example, but this time representing genes through their names, is given below:

```
4 18 s
K D atp8 A R S1 E F nad5 H nad4L T P S2 nad1 L1 rrnL Q
>Nematoda_Trichinella_spiralis
circular
+E +nad1 +K -F -nad5 -H -R -nad4L +T -P +S1 +rrnL +Q +D +atp8 +S2 +L1 +A
>Arthropoda_Anopheles_quadrimaculatus
circular
+K +D +atp8 +R +A -S1 +E -F -nad5 -H -nad4L +T -P +S2 -nad1 -L1 -rrnL -Q
>Arthropoda_Apis_mellifera
circular
+D +K +atp8 -R -F -nad5 -H -nad4L +T -P +S2 -nad1 -L1 -rrnL +E +S1 +Q +A
>Hemichordata_Balanoglossus_carnosus
circular
-S2 +D +K +atp8 +R +nad4L +H +S1 +nad5 +E +T -P +F +rrnL +L1 +nad1 -Q -A
```

In the first line of the file, the literal format was indicated by the [*s*] option and, in the second line, the gene names were specified. For additional examples, check the files in the directory "data/input".

3.2 Phylogenetic trees — Newick format

The Newick format is a popular tree representation format, commonly used by phylogenetic reconstruction programs to represent a phylogenetic tree. In this project we adopted a simplified version of the Newick format, for which all trees are also valid Newick trees, but the converse is not always true.

In our simplified format, only the leaves of a tree, where we have the known species, are labeled. The label can be the species name or any text composed by letters, numbers, hyphens or underscores. The label cannot have spaces; use hyphens or underscores instead of spaces.

SCJ suite accepts binary trees only. Given a parent node, its left and right children are represented in the following way:

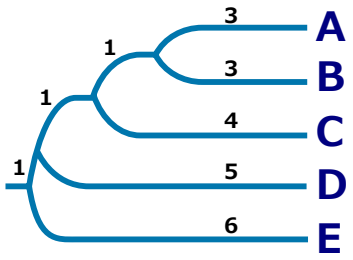
$$(right\_\,child\ :\ weight,\ left\_\,child\ :\ weight).$$

If the child is a leaf, you need to give a label as *right_ child* (or *left_ child*). Otherwise, the structure repeats itself within the argument *right_ child* (or *left_ child*). The edges of the tree have an associated weight, described by a positive integer, and indicating how far apart a genome is from its parent.

This representation cannot contain spaces or tabs. Each line must describe one tree only and, at the end of the line, you need to put a semicolon to indicate the end of the tree representation.

The following example helps clarify the explanation. Notice that all edges should have an associated cost and only the leaves are labeled. For additional examples, check the files in the directory "data/simulatedTrees".

```
((((A:3,B:3):1,C:4):1,D:5):1,E:6);
```



4 *compareTrees* module

SCJ suite is composed by many modules. One of them, *compareTrees*, comprises all processes necessary to compare the expected tree with the resulting tree, inferred by SCJ. This module uses smaller modules and is responsible for executing the entire process, from data preparation to analysis of results, as explained in subsequent sections.

4.1 A brief overview

Based on the information provided by the user, the program: (1) produces the input data or reads them from a file, (2) passes the data to the methods, (3) compares the expected trees to the inferred trees, and (4) stores the analysis results. To use the *compareTrees* module, the user needs to answer questions in three steps:

1. What problem will be solved?
2. What method will solve the problem?
3. What are the input data?

In the first step, the user can choose between solving the Small Parsimony Problem (SPP) or the Big Parsimony Problem (BPP). Depending on the chosen problem, there is a subset of methods available to solve it. In the current version, it is possible to solve the SPP using an adaptation of the Fitch method only, while the BPP can be solved using an heuristic approach (stepwise addition) or an exact approach (branch-and-bound). The last step defines the data, where you can choose either a simulated phylogeny or a genome dataset from file. At each step you can choose only one of the options available.

In the current version, neither the methods to analyze the results nor the format of the output file can be configured by user. Fig. 1 summarizes all steps that the user must define, and the corresponding choices that can be taken. Each of the possible paths in this diagram defines a *usage scenario*.

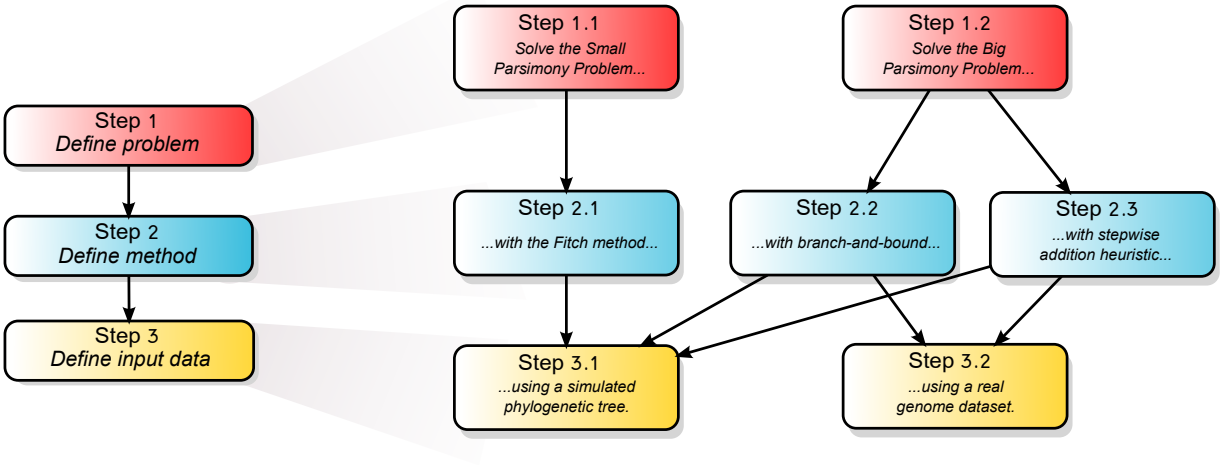


Figure 1: *compareTrees* module — Steps. At each step, the user needs to select only one of the options available. The set of choices of the user composes an *usage scenario*.

4.2 Parameters

The interaction between user and system occurs via parameters. Previously we explained about the decisions that you need to make to define an *usage scenario*. Each choice is characterized by a set of parameters, and specific values are presented in Fig. 2. All parameters and valid values are described in more detail in Table 1.

Some details should be noted, depending on the type of input data. If the input data are real genomes, it is necessary to enter the full pathname of the file that contains the genome dataset, e.g., "C:\scj\data\input\campanulaceae.txt". These genomes must be represented as explained in Section 3.1.

If the input data comes from simulations, it is necessary to enter the full pathname of the tree topologies, e.g., "C:\scj\data\simulatedTrees\size32". In both cases, if OS is Windows, the full pathname must be enclosed in quotes.

In the tree topology directory, each file describes one topology only, according to the simplified Newick format explained in Section 3.2. The files should be named with the following pattern:

`tree[id].nwk`

The argument *id* is a number used as a tree identifier. In the tree topology directory, the tree id should begin with 0 ("tree0.nwk") and be incremented by 1 in each file. The parameter *-iniId* defines from which topology the simulated evolution will start. All topologies with IDs between $iniId$ and $iniId + numTrees - 1$ should exist in the directory, otherwise errors will occur during the simulation.

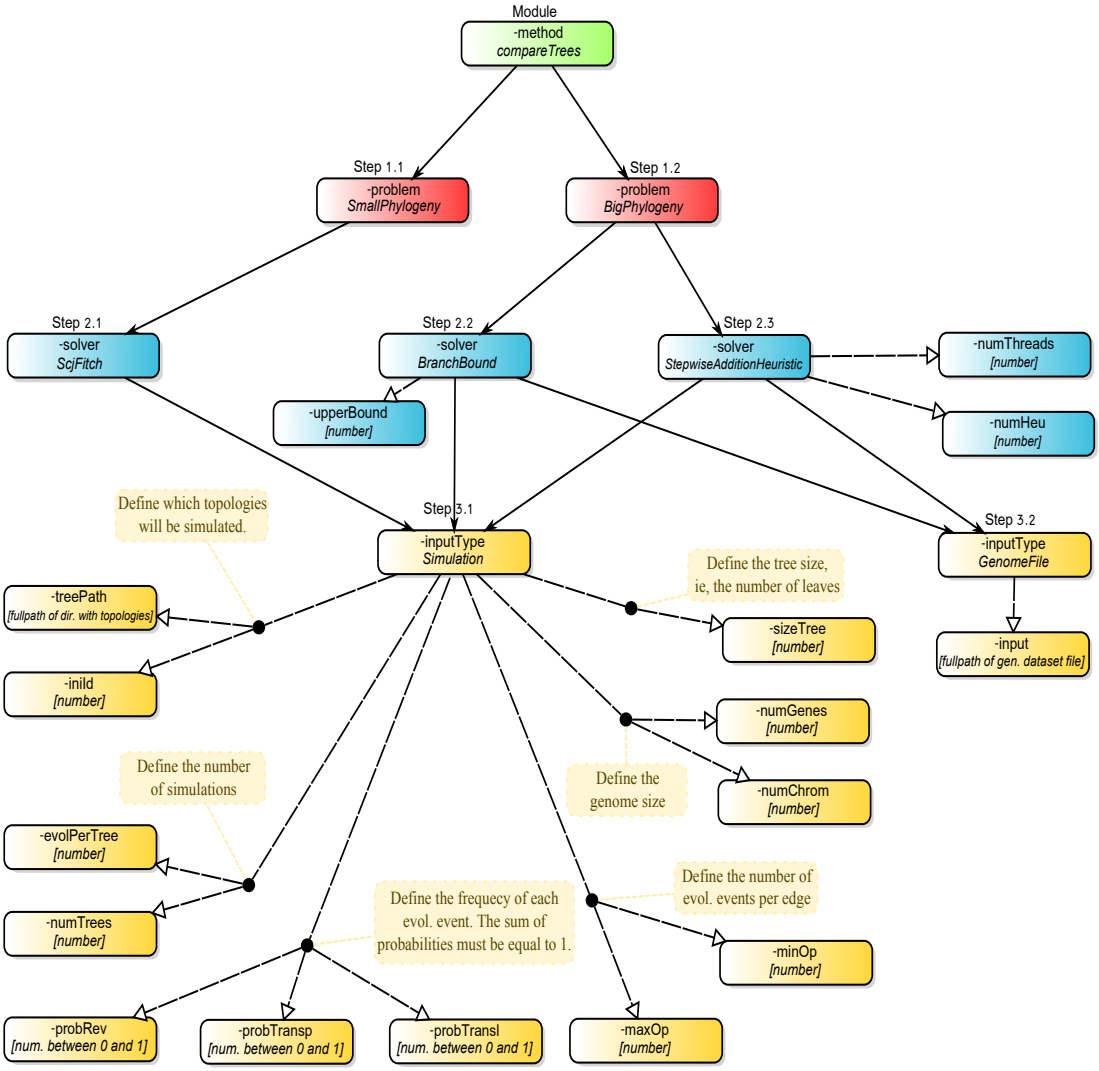


Figure 2: *compareTrees* module — Parameters. In this diagram, each block corresponds to one parameter: the bold text indicates the parameter name, and the text in italics, its value. The relationship between the parameters is represented by arrows. The solid arrows show a possible flow, corresponding to the stages defined in Fig. 1. The dotted arrows indicate the dependency between the parameters, i.e., all parameters that a particular parameter points to with dotted arrows also need to appear in the command.

Table 1: List of all parameters of the module *compareTrees*. The domain of non-numeric types, such as Text, is case sensitive. $\mathbb{N}^*$ denotes all natural numbers except zero.

Parameter	Description	Domain	Steps
-evolPerTree -e	Number of simulated evolutionary events performed on each tree.	$\mathbb{N}^*$	3.1
-iniId	Identifier of the first topology to be simulated.	$\mathbb{N}^*$	3.1
-input -i	Full path to the genome dataset file.	Text	3.2
-inputType	Type of input data.	{Simulation, GenomeFile}	3.1, 3.2
-maxOp	Maximum number of evolutionary events on each edge of tree.	$n \geq [minOp]$	3.1
-method -m	Module to run.	{compareTrees}	—
-minOp	Minimum number of evolutionary events on each edge of tree.	$\mathbb{N}^*$	3.1
-numChrom -c	Number of chromosomes of the root genome.	$\mathbb{N}^*$	3.1
-numGenes -g	Number of genes of the root genome.	$n \geq [numChrom]$	3.1
-numHeu -h	Number of times the heuristic will be performed for each tree.	$\mathbb{N}^*$	2.3
-numThreads	Maximum number of threads to be executed in parallel.	$\mathbb{N}^*$	2.3
-numTrees -t	Amount of topologies to be simulated.	$\mathbb{N}^*$	3.1
-problem	Problem to be resolved.	{SmallPhylogeny, BigPhylogeny}	1.1, 1.2
-probRev -prv	Relative frequency of reversals during evolution on each edge.	$0.0 \leq x \leq 1.0$	3.1
-probTransl -ptl	Relative frequency of translocations during evolution on each edge.	$0.0 \leq x \leq 1.0$	3.1
-probTransp -ptp	Relative frequency of transpositions during evolution on each edge.	$0.0 \leq x \leq 1.0$	3.1
-sizeTree -s	Number of leaves of the tree.	$\mathbb{N}^*$	3.1
-solver	Method to solve the problem.	{StepwiseAdditionHeuristic, BranchBound, ScjFitch}	2.1, 2.2, 2.3
-treePath	Full path to the directory where the topologies used in the simulations are.	Text	3.1
-upperBound -u -ub	Initial upper bound for branch-and-bound.	$\mathbb{N}^*$	2.2

5 Using SCJ suite

In this section we show a few examples of the Java command and its respective output. In the examples, we assume that the OS is Windows and that the file "SCJ-suite.zip" was extracted in the directory "C:\". The examples correspond to the scenarios listed in Table 2.

Table 2: Usage examples of *compareTrees* module. Abbreviations: SPP, Small Parsimony Problem; BPP, Big Parsimony Problem; B&B, branch-and-bound; SA, Stepwise Addition; Sim., Simulation.

Sec.	Problem		Method			Data	
	SPP	BPP	Fitch	B&B	SA	Sim.	Real
5.1	•		•			•	
5.2		•		•		•	
5.3		•		•			•
5.4		•			•	•	
5.5		•			•		•

The Java commands performed in the usage scenarios follow a basic syntax, changing only the command parameters, chosen according to the user's decisions. The basic syntax is as follows:

```
java -jar scj.jar -method compareTrees <parameters of the problem>
                <parameters of the method> <parameters of the input data>
```

The output shows the inferred results and some comparative analysis, and it is stored in a log file, located in the directory "graficos". The log file is named, regardless of the usage scenario, as follows:

```
problem[value]-method[value]-input[value]-[variable parameters and their values].txt
```

If there is a file with the same name, the suffix "-try$n$.txt" is added in the filename, where n is a number not yet used in the composition of similar filenames. Just like the Java command, the log file has a default structure:

1. At the beginning of the log file we will find the values of the input parameters;
2. For each input dataset, the following data are shown:
   (a) Original data and analysis applied to these data;
   (b) Inferred data and analysis applied to these data;
   (c) Summary of the analysis, summary of the results, and the processing time.

5.1 Solving SPP with Fitch using simulated data

Java command

```
java -jar scj.jar -method compareTrees -problem SmallPhylogeny -solver ScjFitch
                  -inputType Simulation -sizeTree 5 -evolPerTree 1 -numTrees 1
                  -numGenes 5 -numChrom 2 -minOp 1 -maxOp 1 -iniId 0 -probRev 0.9
                  -probTransp 0.0 -probTransl 0.1 -treePath "C:\scj\data\simulatedTrees"
```

Output file

```
1  MIN REV  GEN MAX TRP TRL  CHR FOL
2  1   90.0 5   1   0.0 10.0 2   5
3
4  SIMULATION(1/1)        Tue Jan 31 22:54:48 GMT-03:00 2012
5
6  ID  NUM_REV NUM_TRANSL  NUM_TRANSP  BRANCHLEN_SCJ(REAL) BRANCHLEN_SCJ(OPT)  TREE
7  0   7.0     0.0          0.0         14.0                4.0                 (E,(D,(C,(B,A))));
8
9  HEIGHT NODE            %RECONS  FALSO_POS  FALSO_NEG  PARENT          SCJ_DIST_TO_PARENT  GENOME_RECONSTRUCTED     GENOME_ORIGINAL
10 0      A               1        0          0          internal3       0                   [[ 1 -2 ][ -3 4 ][ 5 ]]  [[ 1 -2 ][ -3 4 ][ 5 ]]
11 0      B               1        0          0          internal3       0                   [[ 1 -2 ][ -3 4 ][ 5 ]]  [[ 1 -2 ][ -3 4 ][ 5 ]]
12 0      C               1        0          0          internal2       0                   [[ 1 -2 ][ -3 4 ][ 5 ]]  [[ 1 -2 ][ -3 4 ][ 5 ]]
13 0      D               1        0          0          internal1       2                   [[ 1 2 ][ -3 -4 ][ 5 ]]  [[ 1 2 ][ -3 -4 ][ 5 ]]
14 0      E               1        0          0          Evolution root  0                   [[ 1 2 ][ -3 4 ][ 5 ]]   [[ 1 2 ][ -3 4 ][ 5 ]]
15 1      internal3       0.5      1          1          internal2       0                   [[ 1 -2 ][ -3 4 ][ 5 ]]  [[ -1 -2 ][ -3 4 ][ 5 ]]
16 2      internal2       1        0          0          internal1       2                   [[ 1 -2 ][ -3 4 ][ 5 ]]  [[ 1 -2 ][ -3 4 ][ 5 ]]
17 3      internal1       1        0          0          Evolution root  0                   [[ 1 2 ][ -3 4 ][ 5 ]]   [[ 1 2 ][ -3 4 ][ 5 ]]
18 4      Evolution root  0.5      1          1          -               0                   [[ 1 2 ][ -3 4 ][ 5 ]]   [[ 1 2 ][ 3 4 ][ 5 ]]
19
20 TEMPOS(MS)
21 SIM    INF     SPLIT
22 12     1       0
```

In the output, Lines 1–2 present the parameters values used in the Java command. Line 4 shows the type of input data, the progress of execution, and the time when a specific execution began. Lines 6–7 show the identifier of the simulated topology (ID), the number of evolutionary events for each event type (NUM_REV, NUM_TRANSP, and NUM_TRANSL), and the cost in SCJs of the tree, considering the original ancestral genomes (BRANCHLEN_SCJ(REAL)) and the ancestral genomes inferred by the Fitch method (BRANCHLEN_SCJ(OPT)). Lines 9–18 present the analysis of inferred ancestors for each node of the tree, and also the original (GENOME_ORIGINAL) and reconstructed (GENOME_RECONSTRUCTED) genomes. Lines 20–22 show processing times in milliseconds.

5.2 Solving BPP with B&B using simulated data

Java command

```
java -jar scj.jar -method compareTrees -problem BigPhylogeny -solver BranchBound
             -upperbound 1000 -inputType Simulation -sizeTree 5 -evolPerTree 1 -iniId 0
             -numTrees 1 -numGenes 5 -numChrom 2 -minOp 1 -maxOp 1 -probRev 0.9
             -probTransp 0.0 -probTransl 0.1 -treePath "C:\scj\data\simulatedTrees"
```

Output file

```
1  MIN  REV   GEN  MAX  TRP  TRL   CHR  FOL
2  1    90.0  5    1    0.0  10.0  2    5
3
4  SIMULATION(1/1)        Tue Jan 31 23:14:20 GMT-03:00 2012
5
6  ID  NUM_REV NUM_TRANSL  NUM_TRANSP  BRANCHLEN_SCJ(REAL) BRANCHLEN_SCJ(OPT) TREE
7  0   7.0     0.0         0.0         14.0                4.0                 (E,(D,(C,(B,A))));
8
9  #ID BRANCH  SPLIT   INFER(MS)   SPLIT(MS)   TREE
10 1   4.0     0.5     13          723         (E,((C,D),(B,A)));
11 2   4.0     0.5     13          723         (E,(((B,A),D),C));
12 3   4.0     0       13          723         (E,(((B,A),C),D));
13
14 SPLIT_INFER
15 BEST   WORST   AVG
16 0      0.5     0.33
17
18 BRANCH_LENGTH_INFER
19 BEST   WORST   AVG
20 4      4       4
21
22 TEMPOS(MS)
23 SIM    INF     SPLIT
24 12     39      2170
```

In the output, Lines 1–2 present the parameters values used in the Java command. Line 4 shows the type of input data, the progress of execution, and the time when a specific execution began. Lines 6–7 show the identifier of the simulated topology (ID), the number of evolutionary events for each event type (NUM_REV, NUM_TRANSP, and NUM_TRANSL), and the cost in SCJs of the tree, considering the original ancestral genomes (BRANCHLEN_SCJ(REAL)) and the ancestral genomes inferred by the Fitch method (BRANCHLEN_SCJ(OPT)). Lines 9–12 present the analysis of tree topology (SPLIT) for each optimum tree obtained, i.e., all trees that have the minimum cost, in terms of SCJ (BRANCH). These lines also inform, in the end, the reconstructed topology (TREE). Lines 14–20 summarize the analysis, showing the best and worst splits obtained, and also an average of all values. Lines 22–24 show processing times in milliseconds.

5.3 Solving BPP with B&B using real data

Java command

```
java -jar scj.jar -method compareTrees -problem BigPhylogeny -solver BranchBound
                  -upperbound 1000 -inputType GenomeFile
                  -treePath "C:\scj\data\input\campanulaceae-exemplo.txt"
```

Output file

```
1  ARQ
2  campanulaceae-exemplo
3
4  GENOMEFILE(1/1)        Wed Feb 01 00:20:07 GMT-03:00 2012
5
6  ARQ
7  CAMPANULACEAE-EXEMPLO
8
9  #ID  BRANCH  INFER(MS)  TREE
10 1    70.0    127        (Tobacco,((Codonopsis,(Campanula,Trachelium)),Platycodon));
11
12 BRANCH_LENGTH_INFER
13 BEST   WORST   AVG
14 70     70      70
15
16 TEMPOS(MS)
17 SIM    INF
18 35     127
```

In the output, Lines 1–2 present the genome dataset filename, given by parameter "-treePath" in the Java command. Line 4 shows the type of input data, the progress of execution, and the time when a specific execution began. Lines 6–7 show the genome dataset filename (ARQ) again, since the topology is not known in this case. Lines 9–10 present the optimum trees inferred by SCJ, their brach lengths (BRANCH), the time (in milliseconds) spent (INFER(MS)), and the reconstructed topology in Newick format (TREE). A summary of the best, worst, and average results is given in Lines 12–14. The computational times required to simulate (SIM) and to infer the tree (INF) are shown on Lines 16–18.

5.4 Solving BPP with stepwise addition heuristic using simulated data

Java command

```
java -jar scj.jar -method compareTrees -problem BigPhylogeny
              -solver StepwiseAdditionHeuristic -numHeur 3 -numthreads 3
              -inputType Simulation -sizeTree 5 -evolPerTree 1 -iniId 0
              -numTrees 1 -numGenes 5 -numChrom 2 -minOp 1 -maxOp 1 -probRev 0.9
              -probTransp 0.0 -probTransl 0.1 -treePath "C:\scj\data\simulatedTrees"
```

Output file

```
1  MIN  REV    GEN  MAX  TRP  TRL   CHR  FOL
2  1    90.0   5    1    0.0  10.0  2    5
3
4  SIMULATION(1/1)        Tue Jan 31 23:02:38 GMT-03:00 2012
5
6  ID  NUM_REV  NUM_TRANSL NUM_TRANSP  BRANCHLEN_SCJ(REAL) BRANCHLEN_SCJ(OPT)  TREE
7  0   7.0      0.0        0.0         14.0                4.0                 (E,(D,(C,(B,A))));
8
9  #ID BRANCH  SPLIT   INFER(MS)  SPLIT(MS)   TREE
10 1   4.0     0.5     3          816         ((B,(D,(E,C))),A);
11 2   4.0     0       3          816         ((D,((A,B),C)),E);
12 3   4.0     0.5     3          816         ((B,((E,C),D)),A);
13
14 SPLIT_INFER
15 BEST   WORST   AVG
16 0      0.5     0.33
17
18 BRANCH_LENGTH_INFER
19 BEST   WORST   AVG
20 4      4       4
21
22 TEMPOS(MS)
23 SIM    INF     SPLIT
24 17     10      2450
```

In the output, Lines 1–2 present the parameter values used in the Java command. Line 4 shows the type of input data, the progress of execution, and the time when a specific execution began. Lines 6–7 show the identifier of the simulated topology (ID), the number of evolutionary events for each event type (NUM_REV, NUM_TRANSP, and NUM_TRANSL), and the cost in SCJs of the tree, considering the original ancestral genomes (BRANCHLEN_SCJ(REAL)) and the ancestral genomes inferred by the Fitch method (BRANCHLEN_SCJ(OPT)). Lines 9–12 present the analysis of tree topology (SPLIT) as well as the inferred topology (TREE), for all trees obtained during the execution of the heuristic, repeated according to the number of times given by parameter "-numHeur". Since this is a heuristic, these trees may not reach the minimum cost (BRANCH).

5.5 Solving BPP with stepwise addition heuristic using real data

Java command

```
java -jar scj.jar -method compareTrees -problem BigPhylogeny
          -solver StepwiseAdditionHeuristic -numHeur 5 -numthreads 3
          -inputType GenomeFile -input "C:\scj\data\input\campanulaceae-exemplo.txt"
```

Output file

```
1  ARQ
2  campanulaceae-exemplo
3
4  GENOMEFILE(1/1)        Wed Feb 01 00:16:11 GMT-03:00 2012
5
6  ARQ
7  CAMPANULACEAE-EXEMPLO
8
9  #ID  BRANCH  INFER(MS)  TREE
10 1    70.0    6          ((Campanula,(Codonopsis,(Platycodon,Tobacco))),Trachelium);
11 2    70.0    6          ((Platycodon,(Codonopsis,(Trachelium,Campanula))),Tobacco);
12 3    70.0    6          ((Tobacco,((Trachelium,Campanula),Codonopsis)),Platycodon);
13 4    70.0    6          (((Trachelium,Campanula),(Platycodon,Tobacco)),Codonopsis);
14 5    70.0    6          ((Tobacco,(Codonopsis,(Trachelium,Campanula))),Platycodon);
15
16
17 BRANCH_LENGTH_INFER
18 BEST   WORST   AVG
19 70     70      70
20
21 TEMPOS(MS)
22 SIM    INF
23 36     30
```

In the output, Lines 1–2 present the genome dataset filename, given by parameter "-treePath" in the Java command. Line 4 shows the type of input data, the progress of execution, and the time when a specific execution began. Lines 6–7 show the genome dataset filename (ARQ) only, since the topology is not known in this case. Lines 9–14 present the total cost (BRANCH) and the reconstructed topology (TREE), for all trees obtained during the execution of the heuristic, repeated according to the number of times informed in the parameter "-numHeur". The computational times required to simulate (SIM) and to infer the tree (INF) are shown on Lines 21–23.

References

[1] David Aldous. Stochastic models and descriptive statistics for phylogenetic trees, from Yule to today. *Statistical Science*, 16(1):23–34, 2001.

[2] Priscila Biller. Experiments with the genomic rearrangement operation Single-Cut-or-Join. Master's thesis, Institute of Computing, University of Campinas, 2012. In Portuguese.

[3] Priscila Biller, Pedro Feijão, and João Meidanis. Rearrangement-based phylogeny using the Single-Cut-or-Join operation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2012.

[4] Mary Cosner, Robert Jansen, Bernard Moret, Linda Raubeson, Li Wang, Tandy Warnow, and Stacia Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In *Proceedings of the Conference on Gene Order Dynamics, Comparative Maps, and Multigene Families*, DCAF'2000, pages 99–121, 2000.

[5] Pedro Feijão and João Meidanis. SCJ: a breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8:1318–1329, 2011.

[6] Guido Fritzsch, Martin Schlegel, and Peter Stadler. Alignments of mitochondrial genome arrangements: applications to metazoan phylogeny. *Journal of Theoretical Biology*, 240(4):511–520, 2006.