Softparsmap - Manual

Pär Steffansson

pars at kth dot se

2005-04-22

Revision History

Revision 0.12 2006-03-20 Revised by: PS
Sooce has been replaced by www.jooc.org
Revision 0.11 2005-03-06 Revised by: PS
first official release
Revision 0.1 2003-04-22 Revised by: PS
first draft

Table of Contents

Introduction

As species and their genomes diverge during evolutionary history, the sets of genes and their sequences also diverge. Gene duplication has been proposed as a crucial source of evolutionary innovation in organisms, like Eukaryotes, with small effective population sizes (Ohno, 1970). With duplication, comes initial redundancy, followed by neofunctionalization, subfunctionalization, and most commonly pseudogenization (Lynch, 2001). This differential retention of duplicate genes can result in a different phylogenetic tree for individual gene families than for the species as a whole. When differential parsing of shared ancestral gene and nucleotide polymorphism is added to the picture as well as uncertainty in tree calculation methodologioes, the correlation between the evolutionary history of a gene and a species becomes murky.

In the development of a large scale database for understanding species evolution through the evolution of gene families (Liberles et al., 2001; Roth et al., 2005), it has been necessary to develop a soft parsimony based approach to map gene trees onto species trees.

Disclaimer

No liability for the contents of this document can be accepted. Use the concepts, examples and information at your own risk. There may be errors and inaccuracies, that could be damaging to your system. Proceed with caution, and although this is highly unlikely, the author(s) do not take any responsibility.

Feedback

Feedback is most certainly welcome for this document. Send your additions, comments and criticisms to <david dot liberles at bccs dot uib dot no>.

References

1. A. C. Berglund, P. J. Steffansson, M. J. Betts, and D. A. Liberles, *Optimal gene trees from sequences and species trees using a soft interpretation of parsimony*, Journal of Molecular Evolution (2006).

Overview

Softparsmap is a Java based package implementing the soft parsimony approach presented in [1]. The first part in this section describes the problems that *Softparsmap* solves. The second half gives an overview of *Softparsmap*s features.

Problem

- Given a free gene tree, find where to place the root such that duplications and losses are minimized. Duplications are prioritized before losses.

- Given a rooted gene tree, map the gene tree onto the species tree.
- Removing in-paralogous.

Depending on the model and assumptions this can be solved in many ways.

Solution

Softparsmap is based on the model presented in [1] which is using duplications, losses, and alleles to explain the deviation found between the species tree and the gene tree. *Softparsmap* assumes that species events that is close enough in time such that alleles might survive, is presented in the species tree as uncertainties. Furthermore, genes can only be transfered from one species to another through a speciation event.

With some additional Java programming the package can be adapted to almost any source of data. However, existing interface retrieve *species trees* from NCBI Taxonomy[1], *sequence data* from an XML file, the *gene trees* from one file per family, and the results are written to one or two files per family.

Current available features are

- rooting free gene trees by minimizing duplications and losses while allowing weak edges to be collapsed
- removing in-paralogous while rooting the gene tree or just before the result is saved
- resolve uncertainties by inserting splits from the species tree and using out-groups
- mapping rooted gene trees onto the species trees
- comparing gene trees using splits
- supports the Newick[2] and the Schreiber format for parsing in and out trees
- the package is divided into parts that can be replaced to meet specific needs

Jooc - The Java-based Object Oriented Container

Jooc[3] is a light weight Java-based Object Oriented Container. It assembles Plain Old Java Objects (POJO's) using dependency injection [4] while XML tags in the descriptor use object oriented features. [5] It has no mandatory dependencies outside J2SE 1.4.2 [6] (or later) and is easy to learn.

If you are not familiar with it, make sure you have read the Quick Start found at Jooc[7].

String Templates

A string template is used to extract information from strings using templates, or creating a string by using a template and filling it with data. The class StringTemplate[8] is taking care of this task. The constructor needs a template and two strings which defines the start and the end of all markers in the template. The template can be an empty string, but the strings defining markers must be non empty. Here is an example on how the class can be used.

```
StringTemplate st = new StringTemplate("family{number}.tree", "{", "}");
st.setMarker("{number}", "100");
System.out.println(st.getPaintedTemplate());
st.paintTemplate("family200.tree");
System.out.println(st.getFirstValue("{number}"));
```

Running the code gives following output.

```
family100.tree
200
```

The template in this example is `family{number}.tree` while the only marker is `{number}` since markers start with '`{`' and ends with '`}`'. The second line of code sets the marker to 100 and the next row creates the results and prints it. The fourth row extracts 200 from the value and the next row prints it.

Attributes used to define the start and the end of markers in *Softparsmap* are `markers_begin` and `markers_end`. Some tags in def.xml[9] are using templates and markers in order to extract data from strings and create strings.

Gene and Species Trees

Tree nodes and trees in *Softparsmap* are represented by five classes, Node[10], GeneNode[11], GeneLeaf[12], SpeciesNode[13], and SpeciesLeaf[14]. These classes are linked together in order to create gene and species trees. The class names reflect if the node is an internal node or a leaf. Species leaves and gene leaves are mapped (one-to-many) since each gene sequence (represented by the class GeneLeaf[15]) is found in a species (represented by the class SpeciesLeaf[16]). The species tree is a subtree of the Tree of Life such that every species represented by the species leaves harbor at least one gene sequence found in the gene tree. Before it is possible to map gene nodes onto species nodes in the species tree, every gene leaf has to be added to one species leaf.

A reference to a node has two different meanings depending on the context. Either the reference is used in the context of the node alone, or in the context of a rooted subtree. For instance

```
Set children = someNode.getChildren();
```

returns the children of `someNode`, while

```
Set leaves = someNode.getLeaves();
```

returns the set of leaves in the rooted subtree where `someNode` is the root.

Edge Types

Dependent on the method used when creating gene trees, different types of edges are created. The interface EdgeType[17] defines the properties regarding edge types that this package needs. Every gene node has a reference to an instance of this interface. In order to create your own edge type, extend AbstractEdgeType [18] and the abstract edge tag found in def.xml[19]. For more information see section Create a Tag Instance.

StandardEdgeType [20] is the standard edge type in *Softparsmap* and has following attributes.

- `short_name` - which is the short name used when printing tree tables (should be 1-3 letters long)
- `value_limit` - defines an edge value limit such that any edge value below this limit is defined *weak*, else defined *strong*.
- `divide_value` - determines if the edge value should be divided when making a mid-point re-root

For more information see section Create a Tag Instance.

Printing Trees

There are three methods in the class Node[21] used to print trees to the prompt, `toStringTree()`, `toStringTable()`, and `toStringAll()`. These methods can be used to print species trees as well as gene trees. Each node in the tree has a label and the row in the table with the same label has all available data for that node. If a cell contains '-' it means that the data for that node is not available and if a column is missing it means that the whole tree is missing that data. Here is an example on what a species tree and a rooted gene tree looks like after inferring mutation.

```
       +--(9606)-
(-9347)|
       +-(10090)-
+-------+-------------+--------------------------------+-----------------+
| Label | Class       | Seq                            | Species name    |
+-------+-------------+--------------------------------+-----------------+
| -9347 | SpeciesNode |  -                             | placentals      |
| 10090 | SpeciesLeaf | [20809742, 15126606, 12860621] | transgenic mice |
| 9606  | SpeciesLeaf | [15012045, 16550688]           | man             |
+-------+-------------+--------------------------------+-----------------+
```

This is the species tree and the column `Seq` contains the sequences that exists in a certain species. The column `Species name` is the name of the species.

```
     +----------------(15126606)-
     |
     |              +--(12860621)-
     |       +--(-6)|
(-92)|       |      +--(16550688)-
     +--(-5)|
             |      +--(20809742)-
             +--(-4)|
                    +--(15012045)-
+----------+----------+---------+---------------+------+---------------+-------+
| Label    | Class    | E.v.    | M(g)          | N.i. | SL(g)         | m(g)  |
+----------+----------+---------+---------------+------+---------------+-------+
| -4       | GeneNode | 0.94-UN | [9606, 10090] |  -   | [10090, 9606] | -9347 |
| -5       | GeneNode | 1.0-UN  | [9606, 10090] | D1   | [10090, 9606] | -9347 |
| -6       | GeneNode | 0.92-UN | [9606, 10090] |  -   | [10090, 9606] | -9347 |
| -92      | GeneNode | NaN-UN  |  -            | D1L1 |  -            |  -    |
| 12860621 | GeneLeaf | 1.0-UN  |  -            |  -   | [10090]       | 10090 |
| 15012045 | GeneLeaf | 1.0-UN  |  -            |  -   | [9606]        | 9606  |
| 15126606 | GeneLeaf | 1.0-UN  | [10090]       |  -   | [10090]       | 10090 |
| 16550688 | GeneLeaf | 1.0-UN  |  -            |  -   | [9606]        | 9606  |
| 20809742 | GeneLeaf | 1.0-UN  |  -            |  -   | [10090]       | 10090 |
+----------+----------+---------+---------------+------+---------------+-------+
```

This is the rooted gene tree after mutation has been inferred. The columns and what they stand for are as follows.

- `Label` - is the label found in the tree.
- `Class` - is the name of the class used to represent this node.
- `E.v.` - is the edge value and the short name for the edge type.
- `M(g)` - used to infer mutation. See [1] for more information.

- `N.i.` - contains other node information. At the most it consists of four parts. `D1L2WC` means that the node has one duplication (`D1`), two losses (`L2`), is weak (`W`), and the collapse flag is set to true (`C`).
- `SL(g)` - used to infer mutation. In [1] it is called Z(g).
- `m(g)` - used to infer mutation. See [1] for more information.

Tree Parsers

The interface TreeParser[22] is used to parse in and out gene trees. The Newick and Schreiber format is implemented. It is a redundant interface when it is using streams as well as strings, but since both are used often it is worth it. If you want to create your own parser, extend AbstractTreeParser [23] and the abstract tag found in the file def.xml[24]. For more information see section Create a Tag Instance.

The Newick Format

TreeParserNewick [25] is a parser that supports the Newick[26] format. There are three different versions defined, but you can easily define new ones by extending any of them and change their attributes. Following definitions are taken from the def.xml[27] file. Take a look at the Quick Start for Jooc[28] to find out how attributes and heritage work.

```
<tree_parser did="abstract" eid="instance::linked, interface"
             class="softparsmap.AbstractTreeParser"
             abstract="yes"
             encoding="@@string:=iso-8859-1"
             />

<tree_parser did="newick" eid="abstract"
             class="softparsmap.TreeParserNewick"
             instance_holder="singleton"
             edge_type="@@instance||edge_type::interface:=unknown"
             markers_begin="@@string:={"
             markers_end="@@string:=}"
             recursive_begin="@@string:=("
             recursive_end="@@string:=)"
             recursive_child_separator="@@string:=,"
             marker_edge_value="@@string:={value}"
             marker_leaf_label="@@string:={label}"
             template_node_data="@@string:="
             template_leaf="@@string:={label}"
             before_body="@@string:="
             after_body="@@string:=;"
             />

<tree_parser did="newick_edge_value" eid="newick"
             template_node_data="{value}"
             template_leaf="{label}"
             />

<tree_parser did="newick_edge_value_leaves_too" eid="newick"
             template_node_data="{value}"
             template_leaf="{label}:{value}"
             />
```

The parser named `newick` parse trees without any edge data `"((1, 2), (3, 4))"`, the `newick_edge_value` parser parse trees with internal edge data `"((1,`

`2):100.0, (3, 4):95.0)"`, and the `newick_edge_value_leaves_too` parser parse trees with internal edge values as well as leaf edge data `"((1:100.0, 2:100:0):100.0, (3:100.0, 4:100.0):95.0)"`. Here is the list of important attributes and what they control.

- `encoding` - provides the parser with an encoding to use when parsing in and out gene tree using streams
- `edge_type` - defines which edge type tag instance that will be used
- `before_body` and `after_body` - give the possibility to end and begin the output with constant strings
- `default_node_edge_value` and `default_leaf_edge_value` - define the default edge values in case they are missing
- `recursive_begin` and `recursive_end` - define the start and the end strings of the recursive part of the Newick[29] format. The strings must be unique in the tree string
- `recursive_child_separator` - defines the child separator of the Newick[30] format. The string must be unique in the tree string
- `marker_edge_value` - defines the marker for the edge value
- `marker_leaf_label` - defines the marker for the label
- `template_node_data` - defines the template for edge data for internal nodes
- `template_leaf` - defines the template for labels and data for leaves

The Schreiber Format

This parser is divided up into a tree structure parser and a node converter. This is done in order to make it more flexible by allowing combinations of these two.

As of now there are three structures supported. The most common one is that of type `[[1, 2], [3, [1]]]` which parse the tree structure and call the node converter on every internal node and leaf.

```
<tree_structure_parser did="schreiber" eid="instance::linked, interface"
                       class="softparsmap.TreeStructureParserSchreiber"
                       instance_holder="singleton"
                       before_core="@@string:=["
                       after_core="@@string:=]"
                       left_node_marker="@@string:=["
                       right_node_marker="@@string:=]"
                       node_divider="@@string:=,"
                       child_node_divider="@@string:=,"
                       />
```

Combining this structure parser with the node converter

```
<node_converter did="schreiber_gene_node_label"
                eid="string_template_gene_tree"
                class="softparsmap.StringGeneNodeConverterSchreiberLabel"
                edge_type="@@instance||edge_type::interface:=unknown"
                left_linked_node="@@string:=["
                right_linked_node="@@string:=]"
                />
```

creates the standard Schreiber tree parser

```
<tree_parser did="schreiber_gene_label" eid="dual"
             tree_structure_parser="schreiber"
             node_converter="schreiber_gene_node_label"
             />
```

There are two more structure parsers and six more node converters, creating almost 7*3=21 combinations. Almost, because some combinations is not valid. For more information see the def.xml[31].

In-paralogous

Instances implementing the interface Inparalogous[32] are used to removing in-paralogous. If you want to define your own in-paralog-handler, extend AbstractInparalogous [33] and the abstract tag found in the def.xml[34] file. For more information see section Create a Tag Instance.

InparalogousStandard [35] is the standard procedure to remove in-paralogous in *Softparsmap* and its tag `did="standard"` has following attributes.

- `remove_before_saving` - determines if in-paralogous are removed before saving the final tree or not. The attribute can only be yes or no.
- `remove_while_minimizing_mutation` - determines if in-paralogous are removed while minimizing mutation. The attribute can only be yes or no. The method finds the list by rooting the free gene tree at one edge at the time and then inferring mutation. Before inferring mutation it is possible to remove in-paralogous.

In order to choose the right sequence to replace the rooted subtree, the algorithm is first sorting the sequences using a compare sorting algorithm and then as the last step, the first in the sorted result is chosen. The procedure used to compare two sequences A and B is as follows.

1. if A is complete and B is not, A is preferred,
2. if the item above could not decide then if B is complete and A is not, B is preferred
3. if items above could not decide then if A is longer then B then A is preferred
4. if items above could not decide then if B is longer then A then B is preferred
5. if items above could not decide then if A's GI number is higher then B's GI number, then A is preferred
6. if items above could not decide then if B's GI number is higher then A's GI number, then B is preferred
7. if items above could not decide then A is preferred. It has to choose one

Data Sources

The DataSource[36] interface is responsible of providing the package with data regarding species trees and gene sequences. In order to use your own data you can extend the abstract class AbstractDataSource [37] and the abstract tag found in the def.xml[38] file. For more information see section Create a Tag Instance.

The class DataSourceXmlNcbiTaxonomy [39] is using the NCBI Taxonomy[40] database to extract a species tree for the gene family. The tag is `<data_source did="xml_ncbi_taxonomy" ...>` and contains following attributes.

- `sequence_data` - defines which XML sequence data tag instance that will be used. See section XML Sequence Data below
- `ncbi_taxonomy_names_file` - points to the species name file in NCBI Taxonomy[41] called `names.dmp`
- `ncbi_taxonomy_nodes_file` - points to the species nodes file in NCBI Taxonomy[42] called `nodes.dmp`
- `xml_database_file` - points to your XML database file
- `index_file` - points to the index file. An index file has to be created for the genes found in the XML database. This is done once and by typing

```
java softparsmap.DataSourceXmlNcbiTaxonomy [property file] [data source did]
```

XML Sequence Data

The SequenceDataXml [43] is the class handling the XML detail in the sequence data file. The tag is called `<sequence_data did="xml" ...>` and contains following attributes.

- `file_reading_buffert_size` - determines how many bytes to read from the file in one cycle.
- `main_tag_name` - is the name of the main tag. Under this tag should all data regarding this sequence be placed.
- `sequence_id_tag_name` - is the name of the tag containing the id number for this sequence. These numbers are mapped to the numbers found in the leaves in the gene trees.
- `sequence_tag_name` - is the name of the tag containing the sequence.
- `organism_name_tag_name` - is the name of the tag containing the name of the organism that harbor this sequence. This name is used to map this sequence into the NCBI Taxonomy[44] database.
- `gi_number_tag_name` - is the name of the tag containing the GI number. See NCBI[45] for more information.
- `partial_complete_tag_name` - is the tag containing the value defined in attribute `complete_name` if the sequence is complete, else it is partial.
- `complete_name` - see attribute `partial_complete_tag_name` above.

For all tags but the main and the sequence tag, it is possible to define a template and a marker which is useful if it is necessary to extract data from the tag string. The tag names of the XML detail are case sensitive. For more information see def.xml[46] or the API[47].

In order to implement your own sequence parser you can extend the abstract class AbstractSequenceData [48] . For more information see section Create a Tag Instance and the API[49].

Families and Family Groups

Gene sequences are divided into groups in order to build more reliable gene trees. In *Softparsmap* this kind of group is referred to as a *family*. Families are also divided into groups and they are called *family groups*. A family is represented by the class Family[50] and a family group by the interface FamilyGroup[51]. Instances of this interface are responsible of providing everything needed to perform a task on a family, including the families. In order to create your own family group, extend AbstractFamilyGroup

[52] and the abstract family group tag found in def.xml[53]. For more information see section Create a Tag Instance.

FamilyGroupTreesInFiles [54] retrieves gene trees from files, one tree and family per file. There are four different tags allowed under this tag in order to fill this family group with families. The pair `<include_directory>, <exclude_directory>` are used to include or exclude files from a family group. The attribute `tree_files_directory` is used in these two tags to specify the directory to include or exclude. Last two tags are `<include_group>, <exclude_group>` which are used to include and exclude other groups. These two tags have following attributes.

- `family_group` - defines the did attribute of the family group to include or exclude
- `family_numbers` - can be 'all' to include or exclude all families from the given family group, or comma separated family numbers on those families to include or exclude from the given family group
- `min_family_number` and `max_family_number` - give an upper and a lower limit on those families to include or exclude from the given family group

When including or excluding directories there are two attributes in the family group tag which are used to choose which files to include or exclude. It can be seen as a filter and the attributes are

- `marker_family_number` - is the marker for the file number. It is used to find the number in the file name.
- `template_tree_file_name` - is the template for all files that will be included or excluded from given directories

Here is an example on a few linked family groups.

```
<family_group did="my_super" eid="trees_in_files"
              data_source="my_data_source"
              />

<family_group did="all" eid="my_super">
  <include_directory tree_files_directory="trees/dir_a"/>
  <include_directory tree_files_directory="trees/dir_b"/>
</family_group>

<family_group did="not_yet_valid" eid="my_super">
  <include_group eid="super" family_group="all"
                 family_numbers="123, 456, 789"/>
</family_group>

<family_group did="small_trees" eid="my_super">
  <include_group eid="super" family_group="all"
                 max_number_leaves="20"/>
</family_group>

<family_group did="the_rest" eid="my_super">
  <include_group eid="super" family_group="all"/>
  <exclude_group eid="super" family_group="not_yet_valid"/>
  <exclude_group eid="super" family_group="small_trees"/>
</family_group>
```

The family group with `did="my_super"` contains common attributes. The family group with `did="not_yet_valid"` contains three families with number 123, 456, and 789. Creating a group with small families (`did="small_trees"`) can be useful when you need to test different settings since running the task will not take long. The last family group, `did="the_rest"` makes sure that no family is overlooked. For more information see def.xml[55].

Compute Tasks

The program Compute[56] is used to compute tasks. The usage differ depending on the task and can be printed by typing

```
$ java softparsmap.Compute [property file] [task did]
```

To define your own task, extend the abstract class Compute[57] (or any of its sub classes) and the abstract tag found in def.xml[58]. For more information see section Create a Tag Instance

All tasks extending ComputeFamilyGroup [59] have following usage

```
$ java softparsmap.Compute [property file] [task did]
  [family group did] [ | [family #]]
```

and it is possible to compute all families in the family group or just one by adding the number last. If only one is computed, it is done in verbose mode.

Rooting Gene Trees

The task with `did="root"` roots free gene trees by minimizing duplications and loss while allowing weak edges to be collapsed. In-paralogous can also be removed. The final tree will be written to the file specified by the attribute `template_target` and if there are nodes with more then two children, these nodes are written to a file specified by the attribute `template_target_non_binary`. So the output of this task is one or two files per family. Below is an overview of the algorithm.

1. Find a list of rooted gene trees using the free gene tree. It is possible to remove in-paralogous while finding the root. For details see method SpeciesNode.minimizeMutation(...) in the API[60].
2. From this list the preferred tree is chosen using following criteria
    a. the tree with highest resolution is chosen (as close to binary as possible)
    b. if criteria above could not decide, the tree with highest number of strong edges is chosen
    c. if criteria above could not decide, the tree with the smallest root distance is chosen
    d. if criteria above could not decide, the first tree in the list will be chosen and a warning will be printed

   If you want to choose the preferred tree in a different manner extend the abstract class PreferredTree [61] and change the attribute `preferred_tree` found in this task tag. For more information see section Create a Tag Instance
3. If the attribute `resolve_uncertainties_using_species_splits="yes"` then uncertainties are resolved in the chosen tree by inserting splits from the species tree
4. If the attribute `resolve_uncertainties_using_outgroups="yes"` then uncertainties are resolved by using out-groups in combination with the original tree
5. If the method `removeBeforeSaving()` in the in-paralogous instance defined by the attribute `inparalogous` return true, in-paralogous are removed.
6. If the attribute `resolve_uncertainties_using_outgroups_before_save="yes"` then uncertainties are resolved by using out-groups in combination with the original tree. This is possible because in some cases there might be more uncertainties that the out-group method can resolve after in-paralogous were removed

7. Save final tree to file
8. Save uncertain nodes to file

This tag must be extended to define target attributes.

Mapping a Gene Tree into a Species Tree

The task with `did="map"` will map gene trees onto species trees for a given family group. Duplications and losses are also computed. The result will be written to one file per family. This tag must be extended do define the target attribute.

Compare Gene Trees for Equality

The task with `did="compare_gene_trees"` compare gene trees from two given family groups. It will compare gene trees with the same family number and the pair will be considered rooted as well as free. This task does not have to be extended since no attribute is required.

Gene Family Numbers

The task with `did="family_numbers"` prints the family numbers for a given family group. This task does not have to be extended since no attribute is required.

Create a Tag Instance

There are several parts of *Softparsmap* that can be replace in order to meet more specific needs. The parts are defined by the interfaces in the package. Here is an example on how to create a new parser and install it.

My Parser Code

Let the java file `MyParser.java` contain following lines.

```
import softparsmap.*;
import ooc.*

public class MyParser extends AbstractTreeParser {

  public Node parseIn(String tree) {
    // The code for parsing the string 'tree' goes here.
    // The method returns the root to the parsed gene tree.
  }

  public void parseIn(Node tree, String treeInfo) {
    // The code for parsing in additional information into the
    // tree nodes goes here
  }

  public String parseOut(Node tree) {
    // The code for parsing the gene tree 'tree' to a
    // string goes here.
    // getting an attribute value
    String value = getAttributeString("some_attribute");
    // rest of your code
  }
```

```
}
```

Add the parse in and out code and then compile the class by typing `javac MyParser.java`. If it will not compile, the `CLASSPATH` has probably not been set correctly. More information on how to set the class path is found at Solaris[62] or at Windows[63].

My Property File

Since `MyParser.java` extends the AbstractTreeParser [64] it is a good idea to extend the abstract tag defined in def.xml[65] as well. This is because the abstract tag have some default attributes which are used by the abstract class. Your property file will then look something like this.

```
<?xml version="1.0" encoding="iso-8859-1"?>

<source title="My Project Name">

  <!-- Importing package definitions -->
  <import source="softparsmap/def.xml" source_context="classpath"/>

  <!--
  Your tasks, data sources and family groups can be defined here in
  which you are using your new parser with did="my_parser"
  -->

  <!-- Defining my parser -->
  <tree_parser did="my_parser" eid="abstract"
               class="MyParser"
 some_attribute="@@string:=some value"
 />

</source>
```

The attribute `class` is changed from `AbstractTreeParser` to `MyParser` in order to connect the new code to the tag.

Further Information

The Quick Start Guide[66] will guide you through the configuration and usage of the package. For more information see [1], the API[67], the def.xml[68] file, or the source code[69].

Given a gene node, duplications and loss are computed in class MuNode[70] and pseudo code for this algorithm can be found in following three files: common[71], duplication[72], loss[73].

Notes

1. http://www.ncbi.nlm.nih.gov/Taxonomy/
2. http://evolution.genetics.washington.edu/phylip/newicktree.html
3. http://www.jooc.org
4. http://www.martinfowler.com/articles/injection.html
5. http://en.wikipedia.org/wiki/Object-oriented_programming
6. http://java.sun.com/j2se/

7. http://www.jooc.org
8. ../api/softparsmap/StringTemplate.html
9. ../api/softparsmap/def.xml
10. ../api/softparsmap/Node.html
11. ../api/softparsmap/GeneNode.html
12. ../api/softparsmap/GeneLeaf.html
13. ../api/softparsmap/SpeciesNode.html
14. ../api/softparsmap/SpeciesLeaf.html
15. ../api/softparsmap/GeneLeaf.html
16. ../api/softparsmap/SpeciesLeaf.html
17. ../api/softparsmap/EdgeType.html
18. ../api/softparsmap/AbstractEdgeType.html
19. ../api/softparsmap/def.xml
20. ../api/softparsmap/StandardEdgeType.html
21. ../api/softparsmap/Node.html
22. ../api/softparsmap/TreeParser.html
23. ../api/softparsmap/AbstractTreeParser.html
24. ../api/softparsmap/def.xml
25. ../api/softparsmap/TreeParserNewick.html
26. http://evolution.genetics.washington.edu/phylip/newicktree.html
27. ../api/softparsmap/def.xml
28. http://www.jooc.org
29. http://evolution.genetics.washington.edu/phylip/newicktree.html
30. http://evolution.genetics.washington.edu/phylip/newicktree.html
31. ../api/softparsmap/def.xml
32. ../api/softparsmap/Inparalogous.html
33. ../api/softparsmap/AbstractInparalogous.html
34. ../api/softparsmap/def.xml
35. ../api/softparsmap/InparalogousStandard.html
36. ../api/softparsmap/DataSource.html
37. ../api/softparsmap/AbstractDataSource.html
38. ../api/softparsmap/def.xml
39. ../api/softparsmap/DataSourceXmlNcbiTaxonomy.html
40. http://www.ncbi.nlm.nih.gov/Taxonomy/
41. http://www.ncbi.nlm.nih.gov/Taxonomy/
42. http://www.ncbi.nlm.nih.gov/Taxonomy/
43. ../api/softparsmap/SequenceDataXml.html
44. http://www.ncbi.nlm.nih.gov/Taxonomy/
45. http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html#GInB
46. ../api/softparsmap/def.xml
47. ../api/index.html

48. ../api/softparsmap/AbstractSequenceData.html
49. ../api/index.html
50. ../api/softparsmap/Family.html
51. ../api/softparsmap/FamilyGroup.html
52. ../api/softparsmap/AbstractFamilyGroup.html
53. ../api/softparsmap/def.xml
54. ../api/softparsmap/FamilyGroupTreesInFiles.html
55. ../api/softparsmap/def.xml
56. ../api/softparsmap/Compute.html
57. ../api/softparsmap/Compute.html
58. ../api/softparsmap/def.xml
59. ../api/softparsmap/ComputeFamilyGroup.html
60. ../api/index.html
61. ../api/softparsmap/PreferredTree.html
62. http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/classpath.html
63. http://java.sun.com/j2se/1.3/docs/tooldocs/win32/classpath.html
64. ../api/softparsmap/AbstractTreeParser.html
65. ../api/softparsmap/def.xml
66. ../quick_start_guide/t1.html
67. ../api/index.html
68. ../api/softparsmap/def.xml
69. ../../src/softparsmap
70. ../api/softparsmap/MuNode.html
71. ../pseudo_code/alg_common
72. ../pseudo_code/alg_duplication
73. ../pseudo_code/alg_loss