Softparsmap - Quick Start Guide

Matthew Betts

matthew dot betts at bccs dot uib dot no

2005-04-22

Revision History

Revision 0.1 2003-04-22 Revised by: MB
first draft

Table of Contents

Introduction 3
Quick Start 3
Input Format 4
Further Information 6

Introduction

Many of the methods for producing phylogenetic trees from sequence alignments, for example bayesian estimation (eg. MrBayes, Huelsenbeck and Ronquist 2001) and neighbor joining, produce free trees. Many of the analysis that we want to do on gene trees, for example ancestral sequence and Ka/Ks calculations (eg. TAED, Roth et al, 2005), require or are more understandable when applied to rooted trees. Softparsmap is a java package that uses 'soft parsimony' to root gene trees by mapping them on to a species tree [1].

Softparsmap outputs the rooting that minimizes the number of gene duplication and gene loss events. This parsimony is 'soft' because, prior to rooting, branches with support less than a user-defined threshold are changed so that they agree with the species tree. In-paralogous can be removed either prior to or after rooting, or both, with the most recent complete sequence kept.

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).

Quick Start

Use the accompanying example[1] files by changing to the correct directory:

```
cd example/
```

It is always a good idea to validate the configuration file to make sure it is correct:

```
java ooc.Validate prop.xml
```

It is valid if nothing is returned, else one row per error is printed. Then index the data in the darwin database file (`db.drw`) and the NCBI Taxonomy[2] files (`names.dmp` and `nodes.dmp`):

```
java softparsmap.DataSourceXmlNcbiTaxonomy prop.xml example
```

This index file must be rebuilt after changing the database file or the NCBI files. Then root the gene trees in `./trees/` by mapping them on to the tree of life:

```
java softparsmap.Compute prop.xml root_example unrooted
```

The rooted trees with in-paralogous removed are written in to `./target/`. These trees and their rooting are described in the softparsmap paper [1] and to compute the final map type

```
java softparsmap.Compute prop.xml map_example rooted
```

The files in `target` ending with `.tree` contains the rooted trees in Newick format and the files ending with `.info` contains two rows from the mapping. The first row is the rooted gene tree in Schreiber format and the second row is an array mapping to the internal nodes. If the value is 'D' the corresponding internal node is duplicated.

You can use this on your own data if you replace `db.drw` with your own sequence information, and the trees in `trees/*` with your own, being careful to copy the formats of the example data. More details below.

Input Format

Sequence Information

Sequence information is fed in the form of a darwin database file. Each sequence in this file should contain the following information in the following format:

```
<E>
<ID>sequence_identifier</ID>
<SEQ>peptide_sequence</SEQ>
<ORG>Genus_species</ORG>
<GB>gi_number</GB>
<PAR>completeness</PAR>
</E>
```

Where:

- `sequence_identifier` = an integer that uniquely identifies the sequence in the database file.
- `Genus_species` = the organism that was the source of the sequence
- `gi_number` = GI number of the GenBank entry that contains the coding sequence.
- `completeness` = either `complete`, where the peptide sequence is complete, or `partial`, where it is only a fragment.

In the examples, `sequence_identifier` = the GenBank GI number of the peptide and gi_number = the GenBank GI number of the sequence containing the corresponding gene.

Other formats can be supported by including a new `<sequence_data>` specification in the property file (see manual[3]).

Gene Trees

Gene trees is fed using the Newick[4] format with bootstrap / branch support values represented as internal node labels. Other variations on the newick format can be supported by including a new `<tree_parser>` specification in the property file

(see manual[5]). These trees should be in the directory specified by the family group, and should be called family{identifier}.tree, where `{identifier}` is an integer that uniquely identifies each tree.

Species Trees

Species tree information is fed in the form of the GenBank files `names.dmp` and `nodes.dmp`, which are available at NCBI Taxonomy[6] (or FTP[7]).

Or you can use your own species tree information, as long as it is in the same format as these and that all the species in your gene tree are present.

Property File

The property file describes where the input information is, where the output should go, and what format it is all in. Default formats are available, and are described along in the following. The example property file that accompanies this guide, `prop.xml`, illustrates the following:

- Common definitions can be imported from a separate file. In the example[8], definitions included with the package (see def.xml[9]) are imported:

```
<import source="softparsmap/def.xml" source_context="classpath"/>
```

- Tasks are then defined:

```
<task did="root_example" eid="root"
      inparalogous="new"
      template_target="target/family{number}_rooted.tree"
      template_target_non_binary="target/family{number}_non_binary.info"
      tree_parser_out="newick_bootstrap"
      />

<task did="map_example" eid="map"
template_target="target/family{number}_map.info">
<tree_parser eid="schreiber_gene_label"/>
<tree_parser eid="schreiber_duplication"/>
</task>
```

  The first task called `root_example` inherits from the task `root` (defined in the imported common definitions file def.xml[10]) and computes rooted trees which are put in the directory `target`. The second task called `map_example` find duplications in the rooted trees and writes the result into files located the the target directory.

- Sequences collected together in to a free tree are referred to as a family. Families can be grouped together for processing in the same run. From the example:

```
<family_group did="unrooted" eid="trees_in_files"
              data_source="example"
              template_tree_file_name="family{number}.tree"
              tree_parser_in="newick_bootstrap">
  <include_directory eid="super" tree_files_directory="trees"/>
</family_group>
```

  This specifies a family group called `unrooted`, that inherits from the `trees_in_files` family group (specified in the imported common definitions def.xml[11]), reads sequence data specified by the example data_source (described below), and reads trees from the directory `trees/`. The next section defines a family group that contains the rooted trees:

```
<family_group did="rooted" eid="trees_in_files"
              data_source="example"
              template_tree_file_name="family{number}_rooted.tree"
```

```
                tree_parser_in="newick">
  <include_directory eid="super" tree_files_directory="target"/>
</family_group>
```

This group is used when running the mapping task and other task like comparing the topology of the unrooted and rooted trees.

```
java softparsmap.Compute prop.xml compare_gene_trees unrooted rooted
```

The next sections tell the program where the species tree information and sequence data are kept, and how the sequence data differs from the default format.

```
<sequence_data did="example" eid="xml"
               gi_number_tag_name="GB"
               gi_number_marker="{gi_number}"
               gi_number_template="{gi_number}"
               />

<data_source did="example" eid="xml_ncbi_taxonomy"
             abstract_sequence_data="example"
             ncbi_taxonomy_names_file="names.dmp"
             ncbi_taxonomy_nodes_file="nodes.dmp"
             xml_database_file="db.drw"
      index_file="java_index"
             />
```

- The following lines define weak edges as those with support values less than 0.7:

```
<edge_type did="taed" eid="unknown"
           short_name="UN"
           value_limit="0.7"
           />
```

- The next section describes how to parse input newick trees that have leaves with sequence identifiers as labels and internal nodes with support values as labels.

```
<tree_parser did="newick_bootstrap" eid="newick"
             edge_type="taed"
             template_node_data="{value}"
             template_leaf="{label}"
             />
```

- The final section details what should be done with in-paralogous. `remove_while_minimizing_mutation="yes"` means that in-paralogous will be removed from the tree while the number of gene duplications and losses is minimized. `remove_before_saving` means that they are also removed just before the rooted gene tree is saved.

```
<inparalogous did="new" eid="standard"
              remove_before_saving="yes"
              remove_while_minimizing_mutation="yes"
              />
```

When removing in-paralogous, the complete sequence with the highest GI number of a set of in-paralogous is the one that is kept.

Further Information

For more information see the manual[12], [1], the API[13], the def.xml[14] file, or the source code[15] (local link).

Given a gene node, duplications and loss are computed in class MuNode[16] and pseudo code for this algorithm can be found in following three files: common[17], duplication[18], loss[19].

Notes

1. ../../example
2. http://www.ncbi.nlm.nih.gov/Taxonomy/
3. ../manual/t1.html
4. http://evolution.genetics.washington.edu/phylip/newicktree.html
5. ../manual/t1.html
6. http://www.ncbi.nlm.nih.gov/Taxonomy/
7. ftp://ftp.ncbi.nih.gov/pub/taxonomy/
8. ../../example
9. ../api/softparsmap/def.xml
10. ../api/softparsmap/def.xml
11. ../api/softparsmap/def.xml
12. ../manual/t1.html
13. ../api/index.html
14. ../api/softparsmap/def.xml
15. ../../src/softparsmap
16. ../api/softparsmap/MuNode.html
17. ../pseudo_code/alg_common
18. ../pseudo_code/alg_duplication
19. ../pseudo_code/alg_loss