FAMAP - Manual

(Jan 2014)

FAMAP is a Fast Algorithm for constructing Multiple sequence Alignment and Phylogeny. This manual is designed to introduce the sequence input/output format, tree input/output format and some important functions of the FAMPA program.

1. Sequence I/O format

1.1. Data files

FASTA file

FASTA file format is used for both unaligned and aligned sequence data. The line starting with character “>” contains the description of the sequence, called “label” in the program. Followed by the label line is the sequence of DNA or amino acid.

Special characters in the sequence are as follows. The hyphen “-” is used for representing the indel event. In the FASTA file containing more than one aligned sequence, the dot “.” means code of the loci is the same as that in the first sequence. Example:

```
>Imaginary Sequence A
AGTCGGG---AGTC
>Imaginary Sequence B
.....A.TTT....
```

This means that between the two sequences, there is an insertion/deletion of code TTT and a G/A point mutation. Note: The use of the dot notation is optional. The following representation is also acceptable,

```
>Imaginary Sequence A
AGTCGGG---AGTC
>Imaginary Sequence B
AGTCGAGTTTAGTC
```

Loci file

The aligned sequences can also be stored in a loci file with loci ID and sequence ID displayed. The loci are indexed according to a reference sequence. Consider the following example.

```
>Imaginary Sequence A
AGTCGGG---AGTC
```

```
>Imaginary Sequence B
.....A.TTT....
>Imaginary Sequence C
.....A.......
```

If the reference sequence is “Imaginary Sequence A”, the loci file is as follows:

```
      |0|1|2|
      | | | |
1     |A|A|A|
2     |G|G|G|
3     |T|T|T|
4     |C|C|C|
5     |G|G|G|
6     |G|A|A|
7     |G|G|G|
7     |-|T|-|
7     |-|T|-|
7     |-|T|-|
8     |A|A|A|
9     |G|G|G|
10    |T|T|T|
11    |C|C|C|
```

The filled gaps are not labeled with new loci ID. A fixed space of length 6 is used to write the loci ID.

Warning: Do not output loci file if the number of sequences exceeds the maximum number of characters of a line.

1.2. Data structure

The data structure “DNAdata” is used and is defined in readfile.cpp. It contains two double arrays of char to store the inputted labels and sequences. The total number of sequences read is stored in the variable can be retrieved using the function GetNumSeq().

1.3. Important functions

The functions for I/O of sequence data are defined in readfile.cpp. Important functions are

SequenceFileOpen(char *filename):
For reading one file with FASTA format. The file is allowed to be containing more than one sequence.

SequenceDirectoryOpen(char *directoryname):
For reading all files with extension fasta in a directory.

The aligned sequences can be saved in two formats, FASTA format and a new format that shows the loci ID and the reference sequence. They are defined in the functions Save and SaveSiteID.

2. Tree I/O format

2.1. Tree representations

Root sequence and combined sequence

In the phylogeny program, the first inputted sequence is set as the root of the tree. The combined sequence is initially set as the root sequence. When a new sequence is inputted, it is aligned with the combined sequence. Whenever there is an insertion from the combined sequence to the newly inputted sequence, the root sequence is expanded by filling the gaps with hyphens “-“ and the combined sequence is expanded by filling the gaps with the inserted codes. Example

```
>Previous root sequence
AGTCGGG---AGTC
>Previous combined sequence
AGTCGGGTTTAGTC
>New inputted sequence
ATTCGGGAGTAAAC
```

After alignment, we have

```
>Previous combined sequence
AGTCGGGTTTAGT---C
>New inputted sequence
ATTCGGG---AGTAAAC
```

There is a point mutation from G to T, a deletion of TTT and an insertion of AAA. The loci on the root sequence and combined sequence are updated for insertion only. The result of update is

```
>Updated root sequence
AGTCGGG---AGT---C
>Previous combined sequence
AGTCGGGTTTAGTAAAC
>New inputted sequence
ATTCGGG---AGTAAAC
```

<u>Loci ID</u>

The loci can be indexed according to either the combined sequence or the reference sequence. These two indexing systems are described below.

<u>Combined sequence:</u> Suppose that before the alignment, the loci ID are as follows

```
Loci ID : 12345671118911
                 234  01
Root    : AGTCGGG---AGTC
Combined: AGTCGGGTTTAGTC
```

Here, the IDs 1-11 are assigned to the loci that appear in the root sequence. The loci appear in the combined sequence but not in the root sequence are indexed by 12-14. Suppose that the following new sequence is inputted

```
New     : ATTCGGAAAGAGTC
```

Aligning the new sequence with the combined sequence, an insertion of size 3 is detected. These three newly added loci are indexed by 15-17. The loci ID are assigned as shown below

```
Loci ID : 12345611171118911
                567 234  01
Root    : AGTCGG---G---AGTC
Combined: AGTCGGAAAGTTTAGTC
New     : ATTCGGAAAG---AGTC
```

<u>Reference sequence:</u> Suppose that we have already got the multiple sequence alignment as follows,

```
Seq 1: AGTCGGG---AGT---C
Seq 2: AGTCGGGTTTAGTAAAC
Seq 3: ATTCGGG---AGTAAAC
```

We can set any one of these three sequences as a reference sequence and index the loci according to that chosen reference sequence. The loci that appear on the reference sequence are indexed by consecutive integers starting from 1. The remaining loci are indexed using two numbers, the ID of the loci before the gap and the relative position in the gap. If Seq 1 is chosen as the reference sequence, the six remaining loci are indexed as 7.1, 7.2, 7.3, 10.1, 10.2, 10.3 respectively

```
Loci : 12345677778911111
              ...  00001
              123   ...
                    123
Seq 1: AGTCGGG---AGT---C
Seq 2: AGTCGGGTTTAGTAAAC
Seq 3: ATTCGGG---AGTAAAC
```

Newick tree representation

It is commonly used to represent a tree. Example:

```
             1
            / \
           /   \
          2     3
         / \
        /   \
       4     5
```

The Newick representation is

```
((4,5)2,3)1;
```

The numbers are the species ID. The semi-colon indicates the end of the sentence. Comma is used to separate the siblings of a common parent. One bracket corresponds to one subtree. Extra modifiers can be added after the species ID to represent the weights and the mutations on a node. Here, the following notations are used. For example,

```
4:3<1705>A/C<2654>G/-<2655>T/-
```

The number after the colon is the number of mutations. The number between "<" and ">" is the locus where the mutation occurs. The locus is followed by the mutation. In the above example, 4 is a descendent of 2. From 2 to 4, there is a point mutation from A to C on locus 1705 and there is a deletion of code GT from 2.

Binary tree

The phylogeny generated by the program is always a binary tree. That is, excepting the root, all other nodes have either two or none children. The root and the leaves are labeled by an integer not equal to -1. The ID -1 is reserved for other nodes. The root has one child on the left branch. The leaves have no child. Other nodes have two children.

2.2. Data files

Tree file with topology only

It contains trees in Newick representation separated by semicolon. The file fould_tree.txt contains five trees used in the examples in Robinson and Fould's paper "Comparison of weighted labeled trees". This file shows the topology and the number of mutations without showing the mutations.

When the Newick representation is converted to tree data structure, the functions load_tree and load_ topology defined in phylotree.cpp add empty nodes automatically to make the tree binary.

Warning: The above mentioned functions may produce error if the inputted tree has un-labeled node with only one child. Solving this problem requires merging the problematic nodes. This is not yet implemented in this version of the program.

Tree file

Tree file stores all necessary information for reconstructing one tree.

Line 1 consists of three numbers, number of nodes, length of the combined sequence, and the number of loci on the combined sequence that do not appear on the root sequence.

Line 2 is the Newick representation of the tree

Starting from line 3 is a piece-table showing the indexing of the loci on the combined sequence. Each piece in the piece-table has two numbers, starting locus ID and the length. The piece-table is ended with (-1,0). Example:

```
Loci ID : 12345611171118911
                567 234  01
Root    : AGTCGG---G---AGTC
Combined: AGTCGGAAAGTTTAGTC
```

The piece-table is

```
Start Length
1     6
15    3
7     1
12    3
8     2
10    2
-1    0
```

The piece-table is then followed by the combined sequence. The root sequence can be reconstructed by removing the loci on the combined sequence with loci ID greater than the length of root sequence (written in Line 1).

Dendrogram file

The dendrogram can be plotted with the function print_tree_to_file defined in phylotree.cpp.

2.3. Data structure

The tree is stored in the class familytree which is defined in phylotree.cpp.

Class familytree

SeqComb: Combined sequence

SeqRoot: Root sequence

SeqNew: After aligning the new input sequence with the combined sequence, the new input sequence with inserted gaps is stored in SeqNew, the combined sequence, i.e., SeqComb is updated, and gaps are inserted to SeqRoot.

Loci_id: Given a position, find the corresponding loci ID. Here, the loci are indexed according to the combined sequence system described in Section 2.1 under the title “Loci ID”.

Pos_id: Given a loci ID, find the position on the sequences. Such position is the same for all SeqNew, SeqComb, and SeqRoot.

loci_CRS and loci_CRS2: Given a position, find the corresponding loci ID. Here, the loci are indexed according to the reference sequence system described in Section 2.1 under the title “Loci ID”. loci_CRS stores the part of ID before the dot while loci_CRS2 stores the part of ID after the dot.

Tree data: The phylonode structure par_root has left child pointing to the root of the tree. The root can be changed in the program. Therefore, it is convenient to have a fixed unused node pointing to the root. “Table” is the array of pointers to the labeled phylonode. Given a species ID, one can therefore find the tree node without parsing the whole tree.

3. Sample programs

3.1. Building a tree from the sequence data

This is illustrated in the function test_treebuild.cpp. The file “Tanaka.fas” in FASTA format consists of 672 sequences of mitochondrial DNA collected from Japanese populations. The data was studied in Tanaka, et al. (2004). The sequences can be downloaded from the website phylotree.org. To locate the root, an African sequence is added. The Cambridge reference sequence (CRS) is also added. CRS is the mitochondrial DNA sequence of an individual who claim and is believed to have European ancestry. The loci can therefore be indexed using the CRS. The phylogeny obtained from the program can then be compared with the published results of human mitochondrial DNA analysis.

3.2. Loading an existing tree from a tree file and inputting new sequences

This is demonstrated in test_load_add.cpp. Here, the tree file obtained from Kong, et al (2003) data is read. It contains Newick representation, piece-table of the combined sequence, and the combined sequence. Then, 50 new sequences from Tanaka, et al (2004) are added. The updated dendrogram is outputted.

3.3. Simulating the evolutionary process from a root sequence

This is demonstrated in test_simtree.cpp.

In SimulateOneTree(), the user has to specify the number of species on the tree and input the sequence of the root.

The tree topology, together with the coalescence time of each branch, is simulated using Kingman’s coalescence theory. The function sim_tree_topology(10) generates a phylogeny with 10 species. The species are labeled with 0-9. The species with label 0 is set as the root of the tree. Note that for simplicity, the root is NOT set to the common ancestor which is not labeled.

Mutations are then simulated according to the coalescence time.

The simulated sequences are stored in unaligned.txt. The simulated tree is stored in simulated_tree.txt and sim_dendrogram.txt.

In SimulateOneTreeFromTopology(), the topology is read from a file rather than being simulated. The mutations are then simulated. The examples that are used are in Robinson and Fould’s paper “Comparison of weighted labeled trees”.

3.4. Comparing phylogenies using Robinson and Fould distance

The program testRFdistance.cpp reads five trees from fould_tree.txt and compute the Robinson and Fould distance pair by pair.

The char pointer “str” is used in reading the Newick tree representation, particularly when it is too long to be stored in one single line. When the last delimiters (i.e., the syntax notations, including "():,;<>/") in a line is read, “str” stores the remaining part of the line after the last delimiter.

3.5. Large scale simulations

This is demonstrated in test_simtree.cpp. This can be done using the similar code as in the function SimulateManyTree().

In this example, the number of species is 10. The topology is not given. 10 trees with different topologies are simulated. For each simulated tree, the true alignment is known. The phylogeny is built as if both tree topology and alignment are unknown. The constructed tree is then compared with the true tree. The Robinson and Fould distance is outputted to the text file output.txt.