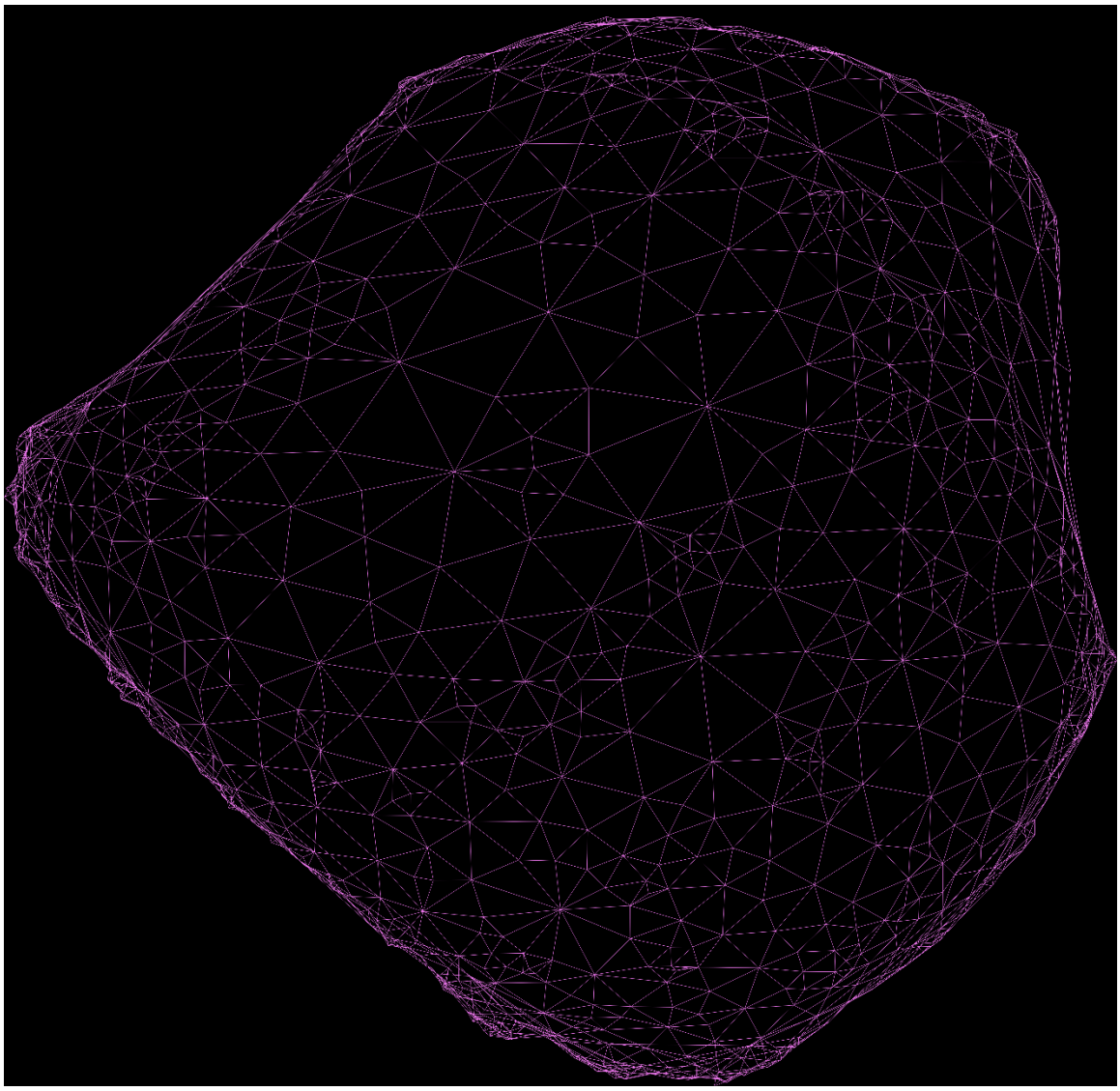


Labeled Eulerian Tour Automation

INSTRUCTION ON THE LABELED EULERIAN TOUR (LET) PROGRAM AND ITS APPLICATION ON DNA ASSEMBLY

Hansi Ji | University of Minnesota-Twin Cities

https://github.umn.edu/jixxx217/Labeled-Eulerian-Tour

Table of Content

OVERVIEW

The *LET_Automation.py* program is written in Python 2.7. This program:

1. generates reference and target sequences;
2. stores the Labeled-Eulerian-tour in a nested list;
3. applies smooth, trim, fork, de-knot, and combine to the list;
4. prints the result in "output.txt".

DATA STRUCTURES

LET

Reference and target sequences can be either randomly generated from **Generate()** (line 89) using **RNG** (line 10) class, or be manually entered through reference.txt (line 590) and target.txt (line 591) using **Reference_given()** (line 212) and **Target_given()** (line 202).

LET is implemented as a nested list, for instance:

self.let = [['CTGAT', [['A', [16, 43]]], [['T', [17, 44]]]], ['AGCCG', [['C', [6]]], [['G', [7]]]], ['GCCCT', [['C', [13, 27]]], [['C', [28]]]], ...]

Layers

There are 5 layers of lists in self.let[index_0][index_1][index_2][index_3][index_4]:

index_0: represents each node in the LET;

index_1: represents each component of a node. 0 is alphabets, 1 is previous edge information, 2 is next edge information;

index_2: represents each previous/next edge with unique alphabets;

index_3: represents each component of a previous/next edge. 0 is alphabet, 1 is edges' label.

Index_4: represents each label of a previous/next edge.

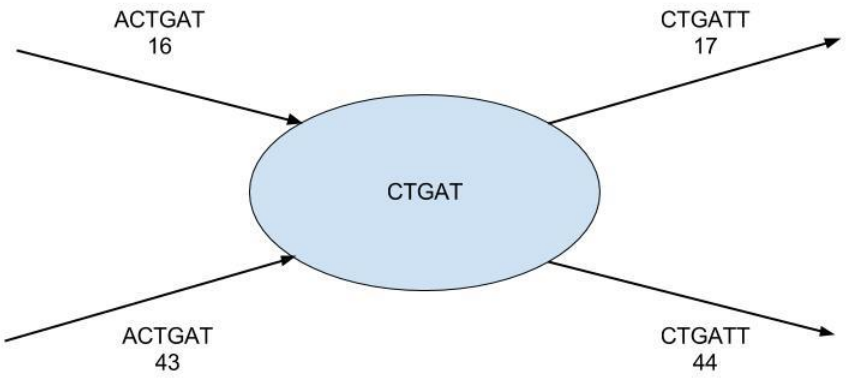
Label Access

In this example, to iterate through the labels of the first previous edge of the first node:

```
for label in self.let[0][1][0][1]:
    do_something(label)
```

Visualization

To visualize the first node:



Graph Representation of
['CTGAT', [['A', [16, 43]]], [['T', [17, 44]]]]

Parameters

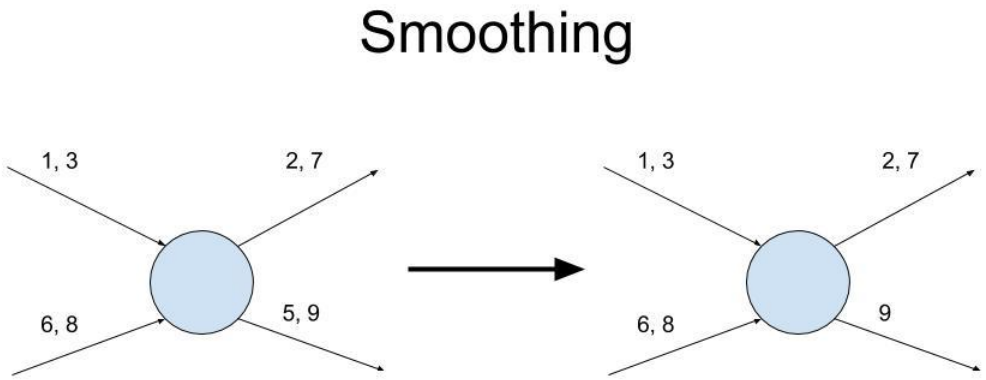
There are 6 parameters and 12 static variables initialized in *def __init__* of GENE class:

Variable Name	Type	Utility	Initial Value
N	Integer	Number of characters of every edge for reference and target sequence.	User input
L	Integer	Window Length of each scan.	User input
M	Integer	Mistakes tolerated within a window length.	User input
P	Double	Probability that a random character randomly mutates.	User input
S	Integer	Seed to Random Number Generator that will affect reference and target sequence.	User input
O	Integer	Operating mode: 0: prints results only; 1: prints results and methods used; 2: prints results, methods used, and input/output of each method. This is time consuming.	User input
rng	RNG object	To generate random reference and target sequence.	Depends on S
LM	Boolean	If the randomly mutated target sequence exceeds the mistake limit within a window.	False
run	Boolean	If a method has modified the LET.	False
done	Boolean	If a sequence has been restored from LET.	False
let	(Nested) List	The Labeled Eulerian Tour implemented in a nested list.	[]
label	Dictionary	Contains initial labels of every edge.	{}
edge	Tuple	Contains initial edges.	()
reference	String	Reference sequence.	''
target	String	Target sequence.	''
start	String	The starting edge.	''
end	String	The ending edge.	''
result	String	Restored sequence from LET.	''

ALGORITHMS

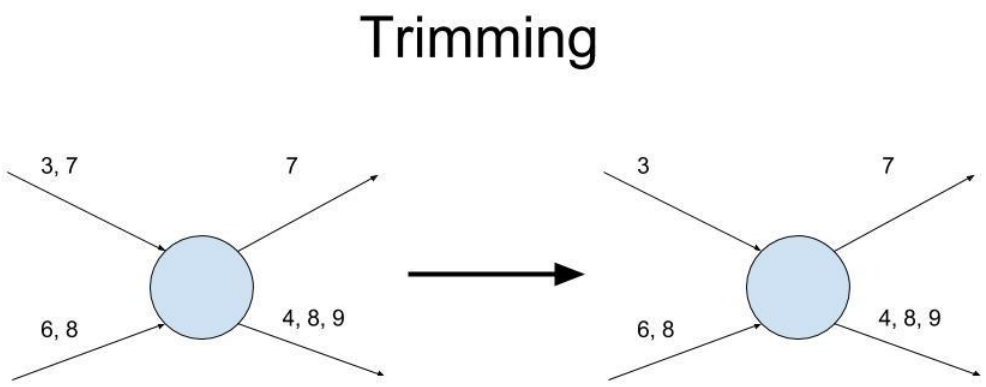
Smooth (line 217)

Smooth does a forward scan (line 220) of every incoming label, and a backward scan (line 237) of every outgoing label. It eliminates a label if it has no previous or next label.



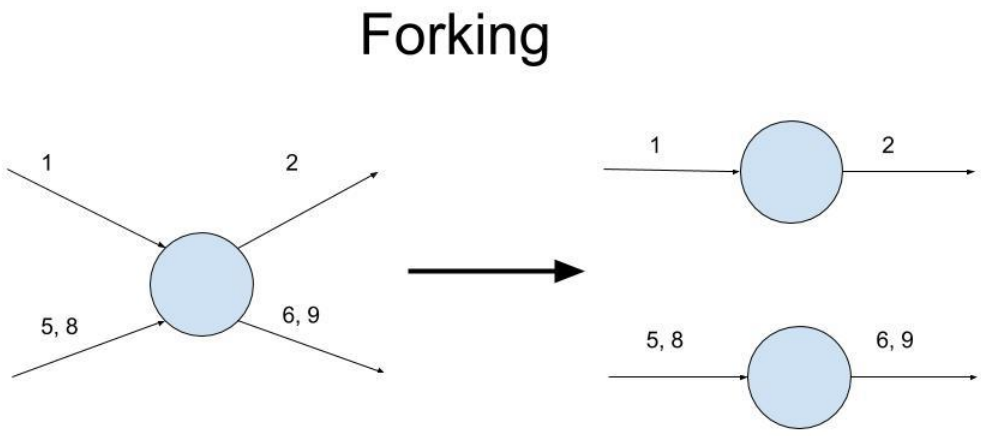
Trim (line 263)

Trim does a global scan on every label of singleton edges (line 269), and eliminates all the same labels from other edges. Exemptions are made for the edge itself, and the same label on its previous or next edge (line 278).



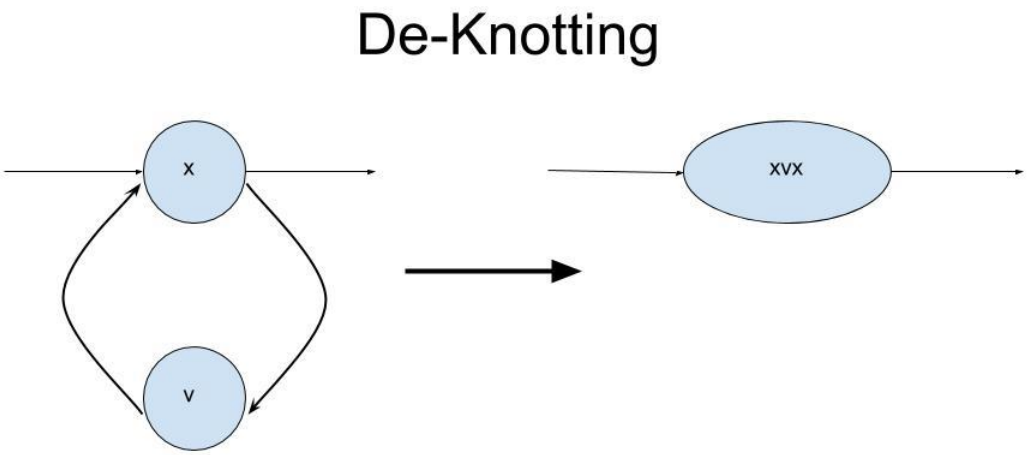
Fork (line 297)

Fork scans every node, if a node can be split into several nodes with one incoming edge and one outgoing edge, it will create new nodes and add them to a list (line 325). Then it will scan every node again to eliminate the ones with new nodes (line 331). Finally, it will add new nodes to the LET.



De-knot (line 385)

De-knot will scan every node to every node with exactly two outgoing edges and two incoming edges (line 395). It will follow both outgoing paths until one path ends with the starting node (line 401). Combining is in every iteration (line 404) and deletion of intermediate nodes (line 417) is in a new iteration.



Combine (line 429)

Combine scans every node and combines nodes with exactly one outgoing edge and one incoming edge together (line 433). The pairing is according to nodes' label on the edge (line 438). Each combine will update a node's characters (line 442) and the label on its outgoing edge (line 443). If a sequence of length of target can be formed, it will terminate all methods and compare the sequence to the target (line 457).