HW2 Construct condensed De Bruijn Graph

Files:

A general python file **hw2.py** is the code I used to process the data and generate the result.

A fasta file **edges.fasta** is the output of edges, with their length and average kmer coverage.

A dot file **edges.dot** is the formatting information of De Bruijn graph and can be visualized with **graphviz** tool.

Results:

1. edges.fasta (sequence omitted)

>edge0 length: 339 coverage: 130.038462

>edge1 length: 513 coverage: 170.473913

>edge2 length: 147 coverage: 233.425532

>edge3 length: 137 coverage: 30.976190

>edge4 length: 67 coverage: 11.500000

>edge5 length: 56 coverage: 9.666667

>edge6 length: 57 coverage: 3.000000

>edge7 length: 339 coverage: 130.038462

>edge8 length: 513 coverage: 170.473913

>edge9 length: 147 coverage: 233.425532

>edge10 length: 137 coverage: 30.976190

>edge11 length: 67 coverage: 11.500000

>edge12 length: 56 coverage: 9.666667

>edge13 length: 57 coverage: 3.000000

There are totally 14 edges assembled from the given fastq file. Noticed that the last 7 edges in fact is the complement sequence of the first 7, there are only 7 contigs I got from my program. The first 3 edges have relative high coverage over 100, while the last 4 is all under 40. Edge3 has a length of 137 which is not short relatively but only has a average kmer coverage of around 30.

2. edges.dot

The formatting information includes all the direction of edges, with their length and coverage.

digragh dbg {

0 -> 2 [label="L:339 \nA\_cvg:130.038462"]

3 -> 1 [label="L:137 \nA\_cvg:30.976190"]

4 -> 1 [label="L:67 \nA\_cvg:11.500000"]

1 -> 9 [label="L:513 \nA\_cvg:170.473913"]

6 -> 2 [label="L:57 \nA\_cvg:3.000000"]

2 -> 8 [label="L:147 \nA\_cvg:233.425532"]

2 -> 12 [label="L:147 \nA\_cvg:233.425532"]

3 -> 4 [label="L:137 \nA\_cvg:30.976190"]

5 -> 9 [label="L:56 \nA\_cvg:9.666667"]

9 -> 7 [label="L:147 \nA\_cvg:233.425532"]

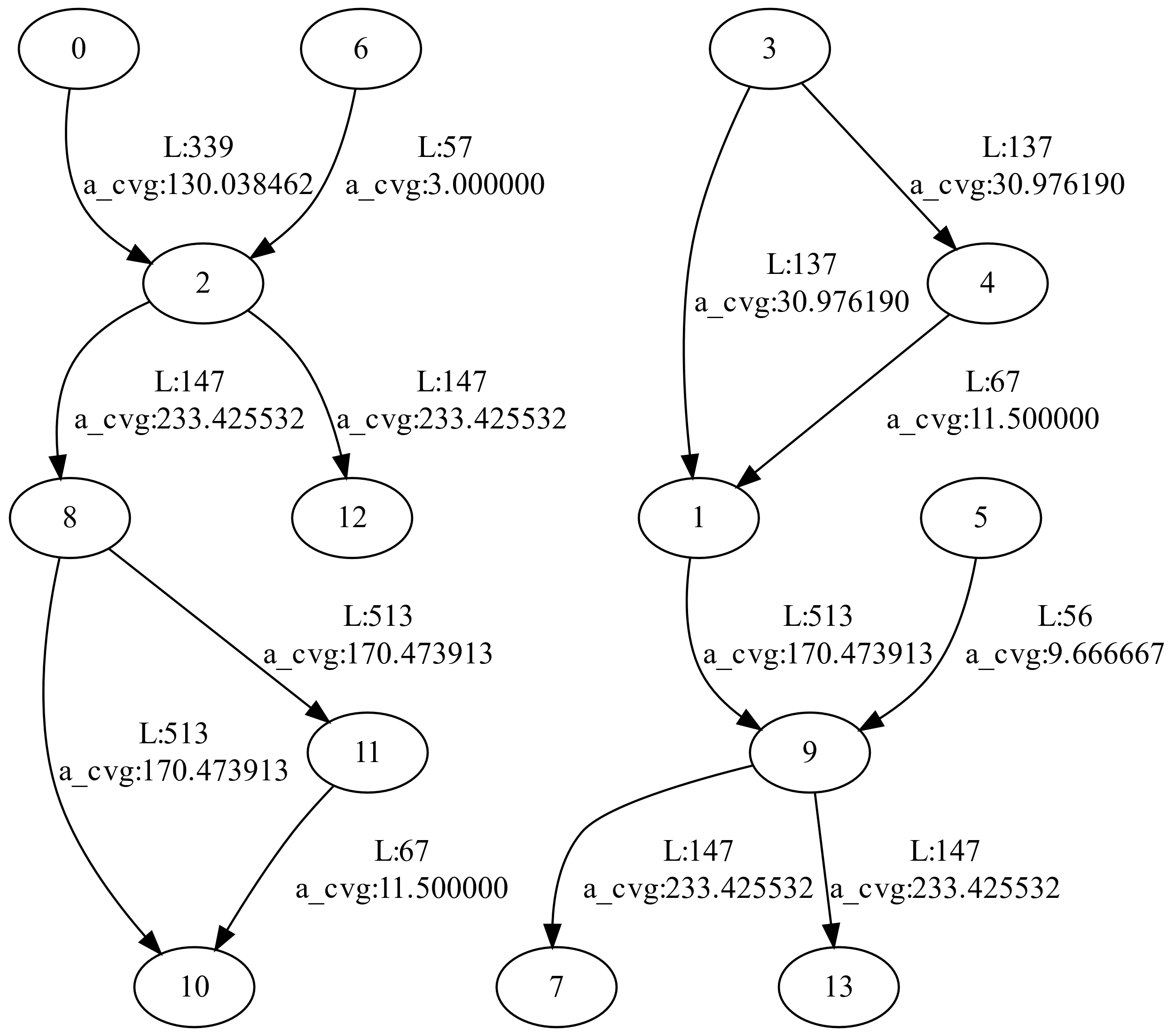
8 -> 10 [label="L:513 \nA\_cvg:170.473913"]

8 -> 11 [label="L:513 \nA\_cvg:170.473913"]

9 -> 13 [label="L:147 \nA\_cvg:233.425532"]

11 -> 10 [label="L:67 \nA\_cvg:11.500000"]

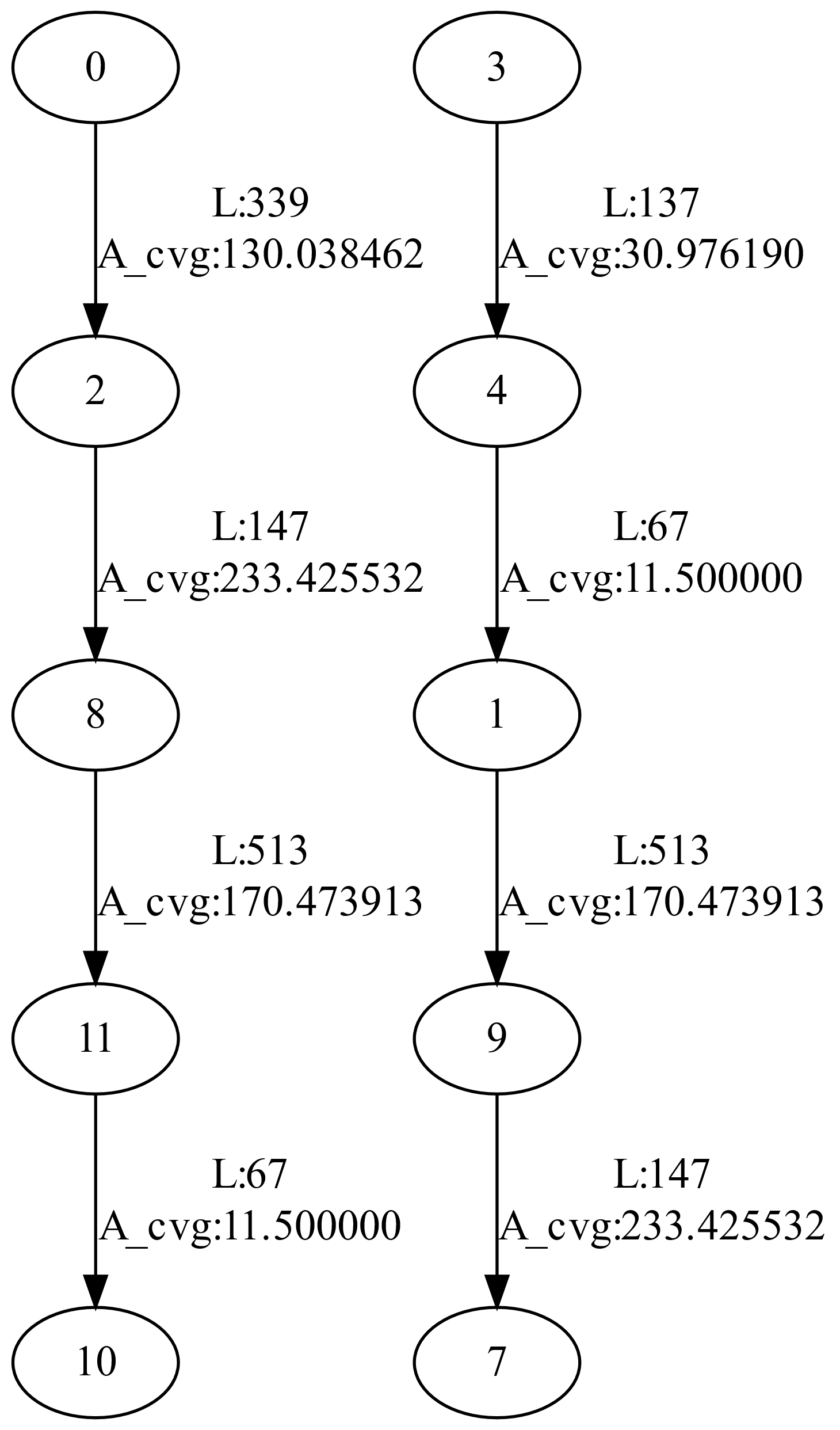
}



The graph is topologically symmetric.

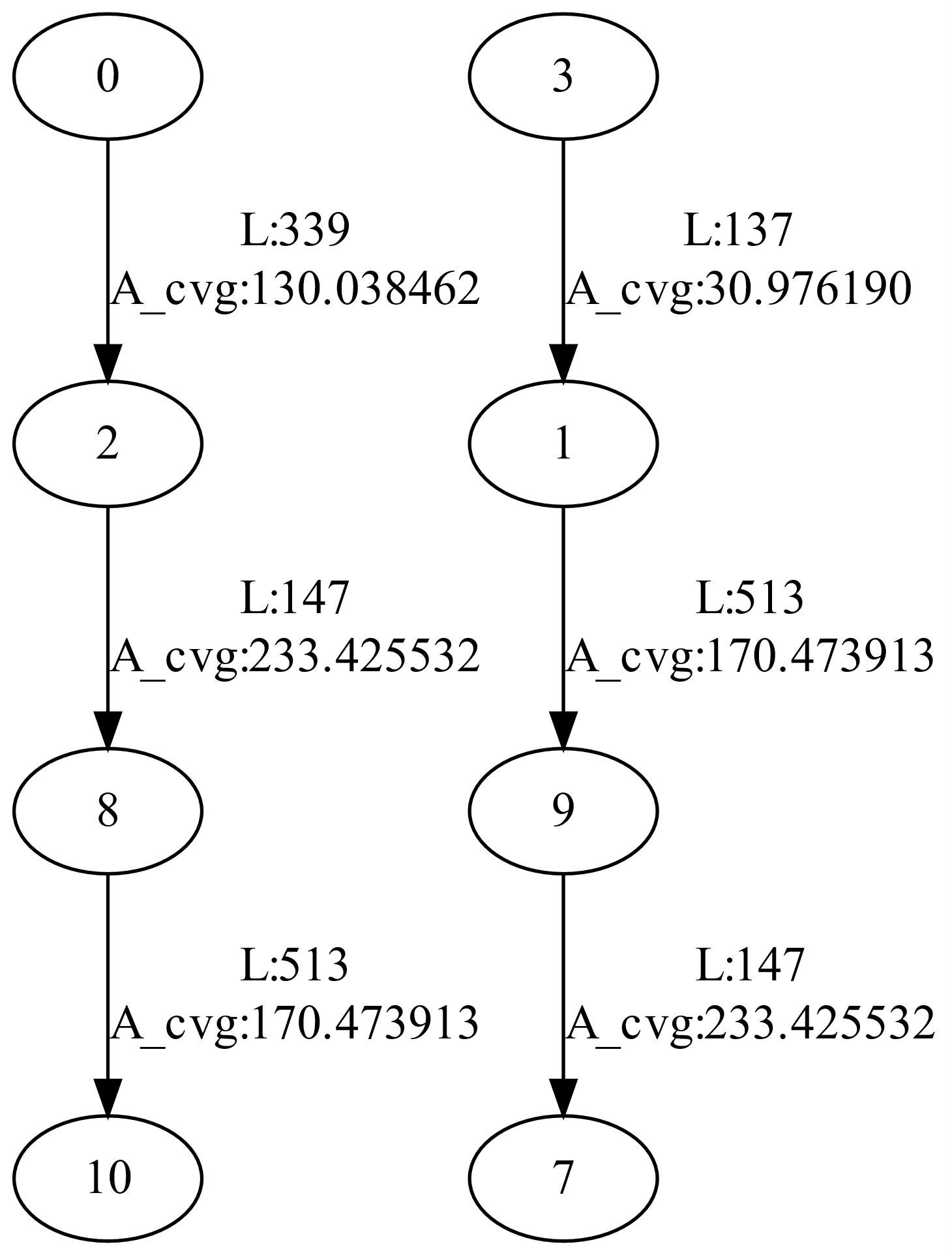
3. Graph simplification

Algorithm1: removing tips with short length and low coverage. If an edge with long length and high coverage is a tip, remove it anyway. whether an edge is a tip depend on whether it is the longest path.



By this algorithm, we always get the longest and still relative complete path in the graph. However, the longest doesn’t always mean correct. Sometimes the relative short pathway could actually exist in real situation like isoform of genes, splice variant etc.

Algorithm 2: remove all the short and low coverage edges beforehand. Edge 3 and 10 is retained here.



This algorithm is quite conservative. It can ensure the quality of later assembly, as all the remaining edges are long and with high coverage. However, it would lose a lot of information of the data, especially those possible and potential pathways but getting a low sequencing quality due to system error.

3. Assembler assessment

1) **ABySS**

Use ABySS to assemble file **s\_6.first1000.fastq**, command in shell as below:

abyss -k 55 s\_6.first1000.fastq -o ./abyss\_result

Get result:

>0 754 122487

TTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAATATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATTACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGAAGTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAGCAATGCCAGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCTGGTGGCGATGATTGAAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATTTTTGCCGAACTTTTGACGGGACTCGCCGCCGCCCAGCCGGGGTTCCCGCTGGCGCAATTGAAAACTTTCGTCGATCAGGAATTTGCCCAAATAAAACATGTCCTGCATGGCATTAGTTTGTTGGGGCAGTGCCCGGATAGCATCAACGCTGCGCTGATTTGCCGTGGCGAGAAAATGTCGATCGCCATTATGGCCGGCGTATTAGAAGCGCGCGGTCACAACGTTACTGTTATCGATCCGGTCGAAAAACTGCTGGCAGTGGGGCATTACCTCGAATCTACCGTCGATATTGCTGAGTCCACCCGCC

2) **minia**

Use minia to assemble the same file, command in shell as below:

minia -in s\_6.first1000.fastq -kmer-size 55 -out minia\_result

Get result:

ACCATGTGATCAGCCGGAATGCGGCTTGCCGCAATACGGCGGGTGGACTCAGCAATATCGACGGTAGATTCGAGGTAATGCCCCACTGCCAGCAGTTTTTCGACCGGATCGATAACAGTAACGTTGTGACCGCGCGCTTCTAATACGCCGGCCATAATGGCGATCGACATTTTCTCGCCACGGCAAATCAGCGCAGCGTTGATGCTATCCGGGCACTGCCCCAACAAACTAATGCCATGCAGGACATGTTTTATTTGGGCAAATTCCTGATCGACGAAAGTTTTCAATTGCGCCAGCGGGAACCCCGGCTGGGCGGCGGCGAGTCCCGTCAAAAGTTCGGCAAAAATACGTTCGGCATCGCTGATATTGGGTAAAGCATCCTGGCCGCTAATGGTTTTTTCAATCATCGCCACCAGGTGGTTGGTGATTTTGGCGGGGGCAGAGAGGACGGTGGCCACCTGCCCCTGCCTGGCATTGCTTTCCAGAATATCGGCAACACGCAGAAAACGTTCTGCATTTGCCACTGATGTACCGCCGAACTTCAACACTCGCATGGTTGTTACCTCGTTACCTTTGGTCGAAAAAAAAAGCCCGCACTGTCAGGTGCGGGCTTTTTTCTGTGTTTCCTGTACGCGTCAGCCCGCACCGTTACCTGTGGTAATGGTGATGGTGGTGGTAATGGTGGTGCTAATGCGTTTCATGGATGTTGTGTACTCTGTAATTTTTATCTGTCTGTGCGCTATGCCTATATTGGTTAAAGTATTTAGTGACCTAAGTCAATAAAATTTTAATTTACTCACGGCAGGTAACCAGTTCAGAAGCTGCTATCAGACACTCTTTTTTTAATCCACACAGAGACATATTGCCCGTTGCAGTCAGAATGAAAAGCT

Compare contigs got from the two assemblers using QUAST.

Genome is choice as first 1K in E.coli, and skip contigs shorter than 50bp.

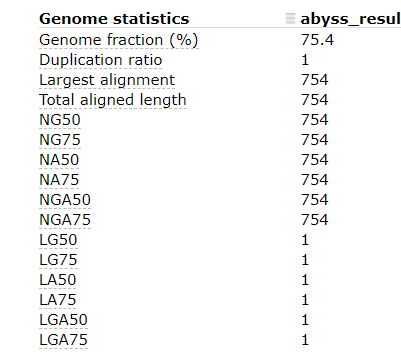
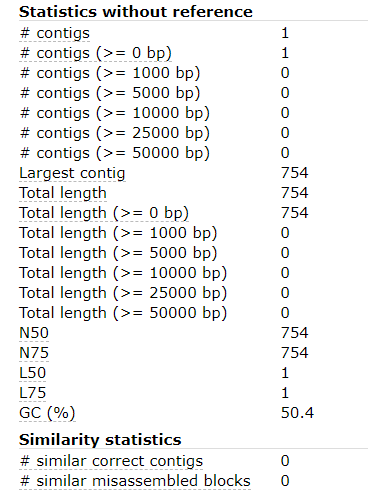
Using ABySS I only got a single assembled sequence and no other contig information was provided. Result showed that only 754bp was assembled in only one contig (75.4%), which is relatively low, and duplication ratio is 1.

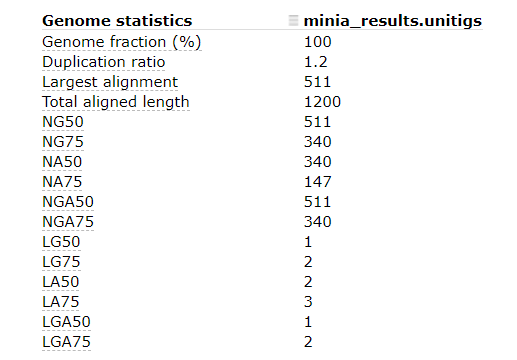
While using minia, results showed that 7 contigs were retrieved, which is consistent to my results.

Moreover, total alignment is 1200 which more than 1k in reference genome, duplication ratio is 1.2 and even 2 contigs are unaligned to the genome.

GC content in contigs generated by the two assembler is different, for ABySS GC% is 50.4% and for minia is 51.86%, with ABySS result more close to the reference (about 50.6%)

We can conclude that ABySS is more conservative and loses some information while minia is getting redundant information after assembly. The former may be more accurate than the later.

 ABySS assembly results:

 minia assembly results:

