Natasha Baas-Thomas

Bio131: Computational Biology

Anna Ritz

May 7, 208

Project Report: Finding a Eulerian Cycle in a Directed Graph

Project:

Eulerian cycles are useful when sequencing genomes. To find out the order of nucleotides in a genome, the DNA is cut up into small pieces that can be sequenced and then these reads need to be reconstructed by relying on overlaps. In an ideal sequencing result, this can be conceptualized in a De Bruijn Graph where the nodes are subsequences, and the directed edges connect subsequences to represent the overlap. A reconstruction of the genome would be complete when we had visited every edge exactly one time. A path that visits each edge one time, and starts and ends at the same node, is called an Eulerian cycle.

In this independent projected, I was motivated to build a code that could find the Eulerian cycle from a dictionary of nodes and edges. The data I used, however, was simplified to just sequentially numbered nodes, instead of genomic subsequences.

Data:

The data were taken form example problems in 'Problem ba3f' of Rosalind.info. The files contain a series of lines that begin with one node, and indicate the edges to which the node is attached to (i.e. "next nodes"). At the end, there is an output line that demonstrates the solution cycle. I simply copied the examples into a .txt file, and no other alterations were made. Test_dict1.txt is a simple directed graph with 10 nodes and 12 edges. Test_dict2.txt is a much larger directed graph with 2002 nodes 2668 edges.

Program:

The first step of the code is to read the .txt files and convert it into a dictionary format where the keys are nodes, and the values are their respective next nodes. This begins by opening the file and splitting each line so that the first number is stored in a list of nodes, and the numbers after the '→' are stored in a list of lists of next nodes. Finally, these two lists are combined into a dictionary, and converted into integers. As such, a key and any of its values together represent an edge.

From this dictionary, we are able to build the Eulerian cycle. The main function begins by randomly selecting a start from the dictionary, and is stored as 'current'. We also define an empty list 'used_edges' that will be updated with two-item lists representing the cycle: [[current, next node], ...]. 'Current' is then tested to see if its next nodes have already been used, by comparing it to 'used_edges'. If there are still unvisited edges, the code randomly selects an unvisited next node. The 'used_edges' and 'current' are subsequently updated. Alternatively, if we reach a node that has no unused edges, the code will search through the 'used_edges', and build a list of nodes that we have visited but still have unused edges. From that list, a node is randomly selected and a different next node is chosen to be the new

'current'. Then, all edges after the new 'current' in the 'used_edges' list are deleted. In other words, the code backs up in the cycle and tries something different at a new node with unvisited edges. Once there are no unused edges, the code outputs 'used_edges' which represents the Eulerian cycle with a random start.

A graph is created in GraphSpace using the list of nodes initially created and the Eulerian path as inputs. The random start is indicated by a yellow star (as opposed to orange circles), and the edges become gradually thicker as the path progresses from the random start.

Results:

I was able to successfully find the Eulerian cycle in test_dict1.txt, and convert it into a graph which is posted on GraphSpace.

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
Node/Edge Dictionary: {0: [3], 1: [0], 2: [1, 6], 3: [2], 4: [2],
5: [4], 6: [5, 8], 7: [9], 8: [7], 9: [6]}

Random start: 2 with possible next node(s) [1, 6]
Cycle: [[2, 6], [6, 8], [8, 7], [7, 9], [9, 6], [6, 5], [5, 4],
[4, 2], [2, 1], [1, 0], [0, 3], [3, 2]]
```

Figure 1. Output from code, using test_dict1.txt. Node/Edge Dictionary was the input for the function to find the Eulerian Cycle.

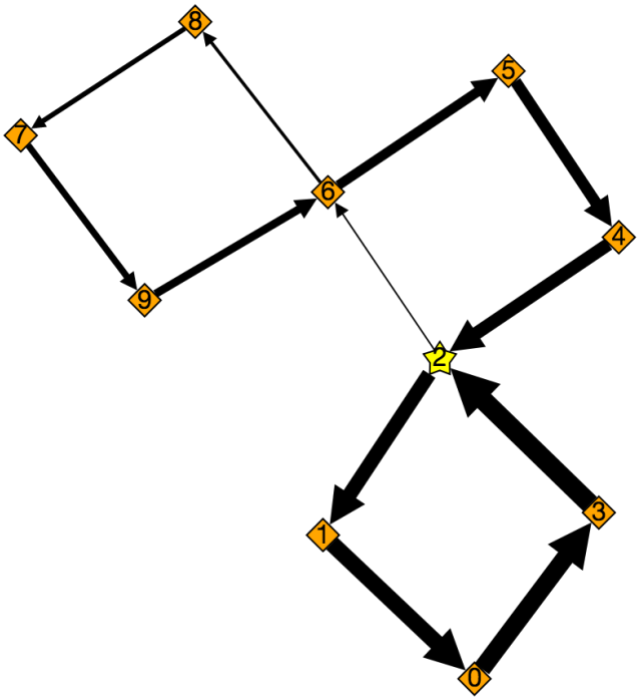

Figure 2. Eulerian Cycle posted in GraphSpace, represented as gradient of increasingly large edges. The yellow star node is indicated as the randomly selected 'beginning' of cycle.

I was unfortunately unable to find the Eulerian cycle for test_dict2.txt due to the size of the data. However, I was able to successfully construct the Node/Edge Dictionary. Using print statements, it was clear that the slowest point in the code was when we had reached a node with all used edges, and the code was searching for a new 'current'. I was also able to use a counter at the while statement on line 56 to stop searching for the Eulerian cycle after a number of attempts, instead of having it stop once all edges had been visited. The output from this modification allowed me to see an incomplete version of the cycle. Therefore, theoretically with enough time and computer capacity, my code could find the cycle for test_dict2.txt.

I think that, in order to shorten the computing time, my code would need to be modified at the point where we reach a node with all used edges. Instead of selecting a new 'current' from the cycle and deleting everything that follows, it would be better to use the new 'current' as a starting point for a sub-path that could then be combined with the main cycle where there was a split.

I think the code could also be expanded in the future to use k-mers or DNA subsequences in order to use the Eulerian cycle, instead of numbers. This alteration would be more useful to help find genomic sequences.