MA446 Project #1 Final Report

Shunfan Du

April 25, 2017

1 Implementations

1.1 Generic Label-Correcting Algorithm

This algorithm will shuffle the list of arcs first, then examine each arc in order to check if any corresponding d needs to be updated to satisfy the optimality conditions. It will keep checking until none of them needs to be updated.

1.2 Modified Label-Correcting Algorithm

All of the modified label-correcting algorithms share these behaviors:

- In the beginning, the source node is added into an empty list.
- Whenever the list is not empty, it will remove a node from the list. Afterward, it will check if the node's outgoing arcs satisfy the optimality conditions, and update corresponding d's. When a d is updated, the algorithm will add its corresponding node into the list.

However, different modified label-correcting algorithms will have different behaviors of adding/removing nodes into/from the list.

1.2.1 Queue (FIFO Algorithm)

The list is implemented using Java's LinkedList.

1.2.2 Stack

The list is implemented using Java's Stack.

1.2.3 Dequeue

The list is implemented using Java's ArrayDeque. It also uses Java's HashSet to track nodes that have been visited before. The algorithm will add nodes that have been visited before into the head of the deque, while it will add nodes that have not been visited before into the tail of the deque. (When I say nodes that

have been visited before, it also means that these nodes have been in the list earlier)

1.2.4 Minimum Distance Label

The list is implemented using Java's ArrayList. The algorithm will always remove the node with the minimum distance label from the list.

2 Comparison

For each problem generator, we will add 3 sources into the *.ss* file: the first node, the middle node, and the last node. All of the algorithms except the generic algorithm will run one time for each source. The generic algorithm will run two times for each source with the list shuffled randomly at each time. The result will be the average of all runs.

2.1 Computational Time (milliseconds)

	Generic	Queue	Stack	Dequeue	MDL
Random4-n.10.0.gr (56 KB)	10	11	162	7	24
Random4-n.11.0.gr (126 KB)	27	36	729	22	38
Random4-n.12.0.gr (256 KB)	38	143	3909	41	124
Random4-n.13.0.gr (544 KB)	57	604	27914	116	479

Summary: Generic $<$ Dequeue $<$ MDL $\approx$ Queue $<$ Stack. Dequeue, MDL, and Queue are slower than Generic probably because it takes some extra time to deal with these datatypes.

2.2 Distance Label Updates

	Generic	Queue	Stack	Dequeue	MDL
Random4-n.10.0.gr (56 KB)	3073	2239	184759	2132	1390
Random4-n.11.0.gr (126 KB)	7202	4745	746333	4757	2745
Random4-n.12.0.gr (256 KB)	14084	9878	2994876	9636	5482
Random4-n.13.0.gr (544 KB)	28894	20981	12147642	20443	10906

Summary: MDL $<$ Dequeue $\approx$ Queue $<$ Generic $<$ Stack. MDL has the least updates probably because the node that has the minimal distance label in the list is more likely to become a part of the shortest path. Therefore, it would be a good idea to update its neighbors' distance labels as soon as possible.

2.3 Arcs Scanned

	Generic	Queue	Stack	Dequeue	MDL
Random4-n.10.0.gr (56 KB)	38229	6977	738233	5863	4096
Random4-n.11.0.gr (126 KB)	95573	14541	2989862	12729	8192
Random4-n.12.0.gr (256 KB)	207530	30713	12002325	25994	16384
Random4-n.13.0.gr (544 KB)	436906	64936	48591781	54984	32768

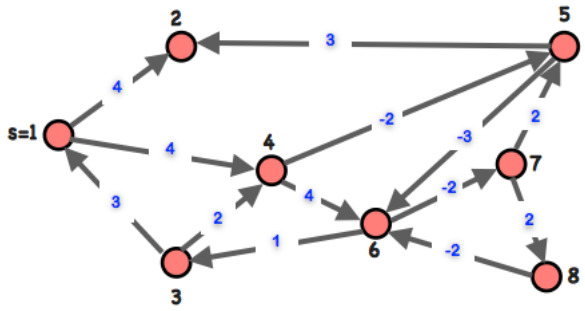
Summary: MDL $<$ Dequeue $<$ Queue $<$ Generic $<$ Stack. The rank is similar to the rank of distance label updates. The arcs scanned are approximately $2 \sim 4$ times the distance updates.

3 Detect and Report Negative Cycles

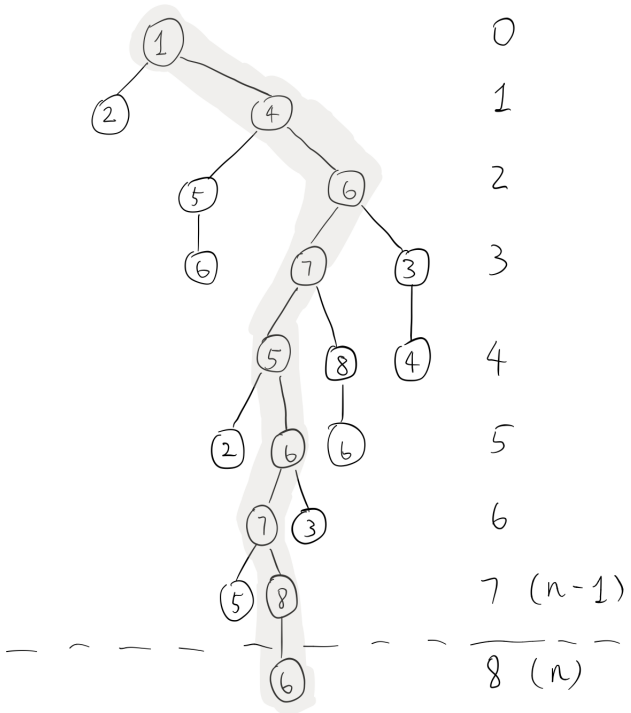
All of the algorithms that I implemented support detecting and reporting negative cycles using the Moore-Bellman-Ford Algorithm.

A path tree will be built whenever a distance label is updated. For example, if we have the following graph:

(Reference: http://algo.epfl.ch/_media/en/courses/2011-2012/algorithmique-cycles-2011a.pdf)



One of the path trees using generic algorithm will look like this:



When a path node is added into the path tree, we check if the depth of the node equals to n. If so, at least one negative cycle exists in the graph. To get one of the negative cycle, we will examine the path from the source node to the node that we just added: $1 \to 4 \to 6 \to 7 \to 5 \to 6 \to 7 \to 8 \to 6$. Since the first number that appeared twice is 6, the first negative cycle that was encountered is $6 \to 7 \to 5$ in this path tree.