MA446 Project #2 Final Report

Shunfan Du

May 18, 2017

1 Assumptions

Here are some assumptions that I made for the input file:

- If a graph has n nodes, the identification number of each node is an integer in the range from 1 to n.
- Each line in the input file does not start with spaces. (Most "*d.rmf" files online does, but I have remove the spaces in my local versions)

2 Implementations

The Java class AbstractAlgorithm implements the abstract concept of all maximum flow algorithms. It has several useful functions:

- void computeExactDistanceLabels(): Compute the exact distance labels.
- Arc getAdmissibleArc(int node): Get one of the node's admissible arcs. Since there is no specification about which admissible arc we should pick, it will get the first admissible arc that appears in the list.
- void updateDistanceLabel(int node): Update the node's distance label.
- void augmentFlow(Arc arc, double value): Augment a flow into the arc and update related arcs' capacity.
- Report generateReport(): Generate the report.

Given the result of the algorithm, here is how I generate the report:

- Maximum flow value = the sum of flows that were augmented through any incoming arc of the sink.
- Arcs with positive flow = Every time a flow is augmented through an arc, I increment the arc's flow value. If an arc has a corresponding reverse arc in the original graph, its flow value will be the arc's value. If an arc does not have a corresponding reverse arc in the original graph, its flow value will be the arc's value minus the reverse arc's value.

- Minimum cut: The program will put all reachable nodes from the source node into a list, and the rest of the nodes into another list. These two lists could indicate the minimum cut.

2.1 Preflow-Push Algorithm

The Java class AbstractPreflowPushAlgorithm implements the abstract concept of the Preflow-Push Algorithm. It includes several functions that are shared by different versions of the Preflow-Push Algorithm:

- void preprocess(): It will call pushFlowsFromSource(), and initialize distance labels and e.
- void pushOrRelabelActiveNode(int node)
- void pushFlowsFromSource()
- void pushFlow(Arc arc)
- void addActiveNodeSafely(int node): Add the node into the active node list only if it's valid.

Moreover, there are several abstract functions, and each version of the Preflow-Push Algorithm will have its own implementation:

- boolean hasActiveNodes(): Check if there are active nodes left
- boolean containsActiveNode(int node)
- void addActiveNode(int node)
- int removeActiveNode()

2.1.1 Queue

The list is implemented using Java's LinkedList.

2.1.2 Stack

The list is implemented using Java's Stack.

2.1.3 Maximum Distance Label Chosen (MDLC)

The list is implemented using Java's ArrayList. The behavior of the function removeActiveNode() has changed significantly: It will remove the active node that has the maximum distance label from the list.

2.2 Shortest Augmenting Path Algorithm (SAP)

The implementation of the shortest augmenting path algorithm is straightforward, it has three basic functions:

- int advance(Arc arc): It takes the admissible arc as the argument instead of the node because if we know the arc, we could easily tell which one is the predecessor.
- int retreat(int node)
- void augment()

3 Comparison

Each algorithm will run 5 times for each input file.

3.1 Size of Problems

	n	m
elist96d	96	528
elist160d	160	912
elist200d	200	1340
elist500d	500	3975
elist640d	640	12608
elist960d	960	9488
elist1440d	1440	22128

3.2 Complexity

	Queue	Stack	MDLC	SAP
Complexity	$O(n^3)$	-	$O(n^2\sqrt{m})$	$O(nm^2)$

3.3 Average Computational Time (milliseconds)

	Queue	Stack	MDLC	SAP
elist96d	16.2	20.8	13.6	5.8
elist160d	8.8	5.2	7.0	4.8
elist200d	10.8	10.2	18.4	7.8
elist500d	21.6	22.4	24.2	35.6
elist640d	290.6	298.2	585.4	290.6
elist960d	1131.2	924.4	1585.2	888.4
elist1440d	3827.0	3218.4	13309.0	2645.4

Summary: MDLC > Queue ≈ Stack > SAP. Everything looks reasonable except MDLC. MDLC runs slower because of the overhead of finding the node that has the maximum distance label in the active node list. I tried to implement MDLC using Java's PriorityQueue, but it didn't make the algorithm run faster.

3.4 Number of Augmentations

	Queue	Stack	MDLC	SAP
elist96d	7025	7357	7131	886
elist160d	5882	6723	5634	4169
elist200d	15336	15642	15657	3105
elist500d	26184	35952	16818	55445
elist640d	196327	211676	183130	118842
elist960d	813160	956129	844304	259133
elist1440d	1990395	2006154	1944579	358284

Summary: Stack > Queue ≈ MDLC > SAP. MDLC has similar augmentations to Stack and Queue because all of the input files have the maximum number of non-saturating pushes n^3 (Stack and Queue) close to $n^2\sqrt{m}$ (MDLC). Moreover, SAP has the least augmentations. However, according to the book, SAP should commonly have more augmentations than a preflow-push algorithm. It's mostly because the augmentations I count in preflow-push algorithm also includes the augmentations where some extra flows are augmented back to the source node, which is not a necessary step and can be improved.

4 Run and Test the Program

4.1 Run

The jar file - project.jar includes all the algorithms.

Usage: java -jar project.jar <Algorithm Code><Input File><Output File>

Algorithm Code:

- Queue
- Stack
- MDLC
- SAP

For example: java -jar project.jar SAP inputs/elist96d.rmf output.txt

4.2 Test

To reproduce the performance test I made, you can use the jar file test.jar

For example: java -jar test.jar

The test results will be printed in the console, like this:

```
Input file: inputs/elist96d.rmf
Algorithm: algorithms.QueuePreflowPushAlgorithm
Maximum flow value: 1.3783999999999998
Number of augmentations: 7025
Average Computational Time (5 runs): 12.0
============================================================
Algorithm: algorithms.StackPreflowPushAlgorithm
Maximum flow value: 1.3784
Number of augmentations: 7357
Average Computational Time (5 runs): 13.2
============================================================
Algorithm: algorithms.MDLCPreflowPushAlgorithm
Maximum flow value: 1.3784
Number of augmentations: 7131
Average Computational Time (5 runs): 13.2
============================================================
Algorithm: algorithms.ShortestAugmentingPathAlgorithm
Maximum flow value: 1.3784
Number of augmentations: 886
Average Computational Time (5 runs): 6.0
============================================================
```