Programming Comps — January 19-26, 2010

1 Background

The maximum multi-commodity flow problem is defined as follows: Given are a capacitated directed graph $G = (N, A, c)$ and $k \in \mathbb{N}$ vectors $d^i \in \mathbb{Q}^{|N|}$ for all $1 \leq i \leq k$. $d^i_u = -1$ means that node $u \in N$ produces commodity i. There is exactly one node u for all i with $d^i_u < 0$, and for it holds $d_u = -1$. Furthermore, it holds that $\sum_{u;d^i_u>0} d^i_u = 1$. $d^i_u = 0$ means that node u is a throughput node which neither produces nor consumes commodity i. Nodes with $d^i_u > 0$ consume commodity i, namely d^i_u of how much is produced of commodity i.

A multi-commodity flow $f^i_{u,v} \geq 0$ specifies how much of each commodity i is routed via each arc $(u, v) \in A$ and satisfies

- $\sum_{i=1}^{k} f^i_{u,v} \leq c_{u,v}$ for all $(u, v) \in E$, and
- $\sum_{u \in N} f^i_{u,v} - \sum_{w \in N} f^i_{v,w} = \lambda^i d^i_v$ for all $i \leq k, \forall_v \in N$

Our objective is to produce and route as much flow as possible, that is, we want to maximize $\sum_i \lambda^i$.

Although solvable in polynomial time (note that fractional flows are allowed), maximum multi-commodity flow problems are very hard in practice. Therefore, approximation algorithms for the problem have been developed. Roughly sketched, these algorithms work by iteratively choosing a commodity, associating a cost with each arc, routing one unit of flow of this commodity via shortest paths, and scaling this flow so that it obeys the capacity constraints. Then, a new commodity is chosen, and so on, until some termination criterion is reached. All flows are then added to one new flow which is scaled down again to satisfy the arc capacity constraints.

2 Your Task

Your task is to implement one part of this approximation algorithm. In particular, you are given a capacitated and weighted directed graph $G = (N, A, c, w)$ and a demand vector $d \in \mathbb{Q}^{|N|}$ (with exactly one negative component which is -1). Your task is to route flow via shortest (according to w) paths from the node u with $d_u = -1$ to all nodes v with demand $d_v > 0$. Then, you are to scale the resulting flow maximally so that it obeys the capacity constraint. The resulting flow is stored in a file.

More specifically: Use Dijkstra's algorithm to compute the shortest path routings. Make a conscious choice regarding the priority queue that you use in your algorithm! Use your knowledge on Dijkstra's algorithm to *efficiently* route the flow and to compute the scaling factor which routes a maximum amount of flow without violating arc capacity constraints.

It is your task to **implement the algorithm as outlined above**. Do not make up a different approach to tackle the problem! If you do, you will fail the exam. The choice of algorithm is fixed, now all that is required is that you fill in the blanks and implement it. The following features must be supported by your implementation:

- Hand in **one** file named collaboration10.tar.gz containing all files of your code, a file named Readme.txt explaining in detail how to compile it, as well as a file named Description.pdf that describes your implementation in detail, including a justification for your choice of language, the structure of your code, and in particular your choice of priority queue and the way how you make use of your understanding of Dijkstra's algorithm to efficiently route flows via shortest paths.
- Your code must compile and run on standard department linux machines with 2GB main memory. If your program does not compile or run on a machine with those specifications, you will not pass the exam. When started, your executable will be provided with two command line arguments: the first contains the file name in which the input graph is given, the second the file name in which the result is to be stored.
- Your executable must be called "approxstep". The program should automatically
  - read in the given file when the program is started,
  - start a cpu-timer (measure cpu user time, **not** clock time!),
  - compute the flow
  - stop the timer and prompt the elapsed time on screen,
  - write the flow into the file, and
  - automatically terminate.
- Use efficient data structures. Save memory and time! In your Description.pdf, explain and justify the data structures you decided to use. Note that your program should be able to solve instances with 100,000 nodes and 15 million edges in roughly 20 CPU seconds (without input and output).

3 Format of Input and Output Files

The weighted and capacitated graph is given as follows. In the first line of the file is given the natural number n of nodes on the graph. In the next line follows a sequence of $n+1$ numbers which give the start index of all arcs going out of each node in the next line, whereby the n+first number gives the total number of arcs. In the third line follows a list of the target nodes of all arcs. In the fourth line follow in the same order the (natural) capacities of all arcs, and in

the fifth line the (rational) weights of all arcs. The final line contains n values representing the demand of each node, where -1 indicates the source of the flow, and the rest of the demands sum to 1.

Your output file should copy the first three lines of the input file. In the fourth line you the write a sequence of rational numbers giving the flow on each arc.

4 Evaluation Criteria

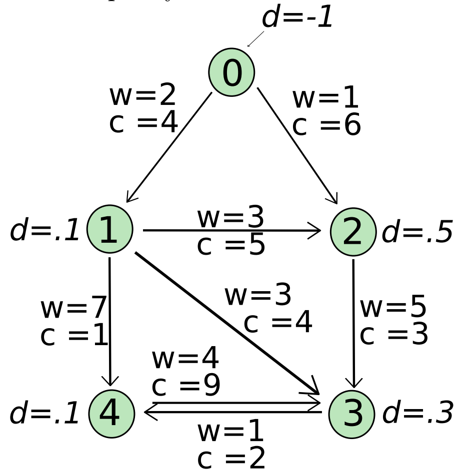
Your implementation will be evaluated based on **all** of the following criteria:

- Appropriate choice of language. You **must** choose a programming language that is best for the task! Ease of programming and maintainability is a concern, but the main focus in this problem is speed.
- Structure, readability, and maintainability of your code.
- Correctness, memory efficiency, and speed.

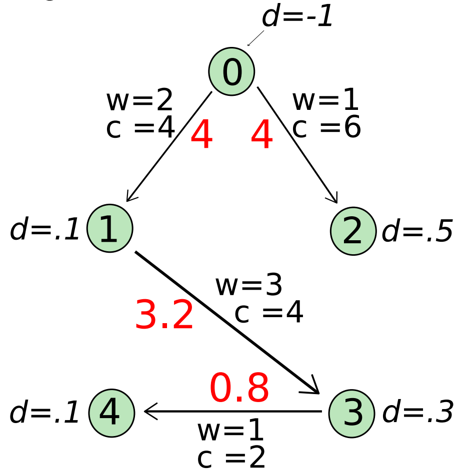
It is important that your work passes a critical threshold with respect to all criteria. If you cannot complete all specifications, focus on the main algorithm and functionality that is required. A super-efficient code that is poorly documented and hardly maintainable will be regarded as a failure. So will a software engineering masterpiece that does not work as specified or that does not perform well. Definitely do not code in assembler or shell scripts! And now: **Good luck!**

5 Appendix

In the following example graph, d is demand, w is the weight of an edge, and c is the capacity.



Performing the algorithm above, lambda is 8 and the flows are shown in red. Edges with a flow of 0 have been removed for clarity.



The corresponding input and output files are as follows:

example.in:

```
5
0 2 5 6 7 8
1 2 2 3 4 3 4 3
4 6 5 4 1 3 2 9
2 1 3 3 7 5 1 4
-1 0.1 0.5 0.3 0.1
```

example.out:

```
5
0 2 5 6 7 8
1 2 2 3 4 3 4 3
4 4 0 3.2 0 0 0.8 0
```