Maximum flow problem

CE 377K

March 3, 2015

Informal evaluation results

- 2 slow, 16 OK, 2 fast
- Most unclear topics: max-flow/min-cut, WHAT WILL BE ON THE MIDTERM?
- Most helpful things: review at start of class, posting lecture slides, notes, algorithm demonstrations
- Things to improve: making expectations more clear, more homework, more real-world applications

Practice midterm will be posted later this week.

Course project

Find a group of approximately 3 students, choose a topic.

Each group should formulate and solve a real-world problem which involves optimization.

In the last week of class, each group will present their project to the class.

A final report will also be due.

By the end of March, each group should give me a project abstract — but I recommend starting earlier.

HW 2 due Thursday

REVIEW

Max flow problem

Residual graph

Augmenting path algorithm

Max flow-min cut duality

Proof of augmenting path algo AND max flow-min cut

COMPLEXITY OF AUGMENTING PATH ALGORITHM

COMPLEXITY

The other issue is proving that the augmenting path algorithm must terminate.

This is easy to do if we assume that the capacities are integers.

(If they are fractional, multiply by a sufficiently large number.)

Each iteration adds at least 1 to b, so b will reach its maximum value b^* in a finite number of iterations.

If U is the largest capacity on any link, then the capacity of the min cut is bounded by nU. (Look at a cut where R is just the source.)

So, the augmenting path algorithm requires $O(nU)$ iterations.

Each iteration must construct a residual network ($O(m)$ steps), and identify a path with positive capacity.

The basic search algorithm for finding a path also has $O(m)$ steps.

So, the augmenting path algorithm in all requires $O(nmU)$ steps.

What could possibly go wrong? This is especially a problem when we have fractional capacities.

A FASTER ALGORITHM

Capacity scaling

We can make the augmenting path algorithm faster if we require that it only choose paths with a high capacity.

One choice is to force it to find the path in the residual network with the *highest* bottleneck capacity.

However, this problem is a bit lengthy to solve. (It is essentially requires solving a shortest path problem, $O(n^2)$ at each step.)

So, instead of augmenting flow along a path of *maximum* capacity, we choose to augment flow along a path of *sufficiently high* capacity.

Let Δ be a "threshold" capacity value.

In the augmenting path algorithm, we will ignore any link whose capacity is less than Δ.

Finding such a path (or determining none exists) is nothing more than the basic search algorithm, requiring $O(m)$ steps.

The idea of capacity scaling is to start with a high value of Δ, then decrease it whenever there are no paths in the residual network with capacity Δ.

Algorithm

1. Initialize Δ to be the greatest power of 2 which does not exceed U.
2. Initialize the flow with $\mathbf{x} \leftarrow \mathbf{0}$, $b \leftarrow 0$.
3. Construct the residual network $\mathcal{R}(\mathbf{x})$ corresponding to flows $\mathbf{x}$.
4. Identify a path π in $\mathcal{R}$ connecting s and t which has at least Δ capacity on each of its arcs. (If no such path exists, divide Δ by 2 and try again. If Δ is less than 1, terminate.)
5. Find the minimum capacity u^* of all of the arcs in π (whether forward or reverse).
6. For each forward arc (i, j) in π^k, $x_{ij} \leftarrow x_{ij} + u^*$. For each reverse arc (j, i) in π^k, $x_{ij} \leftarrow x_{ij} - u^*$.
7. Increase b by u^*, and return to step 2.

Example

Correctness

If at any point the algorithm becomes stuck, Δ will decrease.

When $\Delta = 1$ there is no difference between the capacity scaling and regular augmenting path algorithms, and we know the regular algorithm is correct.

Complexity

Look at the flow in the network just before Δ is reduced. At this point there is no path from s to t in the residual network, using links of capacity more than U.

Therefore the only cut links in the residual network have capacity no more than Δ, and the remaining flow which can be sent cannot be increased by more than $m\Delta$.

Since the next steps will use a threshold of $\Delta/2$, this means that at most $2m$ steps will be taken before Δ is reduced again.

Furthermore, in $O(\log U)$ steps Δ will be reduced to 1.

So, in all this version of the algorithm only requires $O(m \log U)$ reductions of $\Delta_¡$ or $O(m^2 \log U)$ steps in all.