6.854 Dan Ports
Advanced Algorithms drkp@mit.edu
2004/10/06 Collaborators: {sarahl, [redacted]}

Problem Set 4

Problem 1:

a) We show that a maximum flow can be found on a unit network using $O(\sqrt{n})$ blocking flow steps. Note first that the residual network of a unit network is also a unit network: since the flow found has unit values on all edges, subtracting the flow from the network also gives unit capacities, and the network still has either in-degree or out-degree 1 at every vertex. At most one unit of the residual flow can pass through any vertex: since the edges have unit capacity, the degree constraint means that either only one unit of flow can enter the vertex or only one can leave it, and any flow entering the vertex must also leave it. So, letting f be the value of the flow after a blocking flow set, and f_{max} be the maximum flow, the maximum augmenting flow in the residual network is $f_{max} - f$. This means that there are paths from the source to the sink in the residual network that give a total flow of $f_{max} - f$; since the edges have unit capacity, this is the number of paths that must exist. Since no vertex can be on more than one path (otherwise it would need to pass more than one unit of flow), the length of an augmenting path cannot be longer than $\frac{n}{f_{max}-f}$.

Now suppose $\sqrt{n}$ blocking flow steps have already been computed. Then the shortest path from the source to the sink must have at ~~most~~ least $\sqrt{n}$ edges. This is a lower bound, so it must be less or equal to than our upper bound:

$$\sqrt{n} \leq \frac{n}{f_{max} - f}$$

$$f_{max} - f \leq \sqrt{n}$$

So the length of the longest augmenting path is at most $\frac{n}{f_{max}-f} = \sqrt{n}$, so another $\sqrt{n}$ blocking flow steps will complete the algorithm. Combined with the initial $\sqrt{n}$ blocking flow steps, this gives us a bound of $O(\sqrt{n})$ blocking flow computations to find the maximum flow.

b) Let d be the distance from the source to the sink in an arbitrary unit-capacity flow network. We show that the maximum flow is $O\left(\frac{n^2}{d^2}\right)$. Consider the admissible graph. Note that the admissible graph must have d layers, since the distance from the source to the sink is d.

Observe that there must be two adjacent layers in the admissible graph that do not contain more than $\frac{2n}{d}$ vertices. To see this, note that no more than half of the d layers can contain more than $\frac{2n}{d}$ vertices (otherwise there would need to be more than $\frac{d}{2}\frac{2n}{d} = n$ vertices in the graph). So two layers with at most $\frac{2n}{d}$ vertices must be adjacent. Every flow from the source to the sink must pass through one vertex in the first of these two adjacent layers, then through a vertex in the next. Thus, the maximum flow is bounded above by the capacity of the edges between these two adjacent layers. Since there are $\frac{2n}{d}$ vertices in each layer, the total number of edges connecting the two layers can be at most $\left(\frac{2n}{d}\right)^2 = \frac{4n^2}{d^2}$. The edges all have unit capacity, so this is a bound on the capacity of the cut between the two layers. Hence, the maximum flow is at most $\frac{4n^2}{d^2} = O\left(\frac{n^2}{d^2}\right)$.

We use this bound to show that the number of blocking flow computations required to compute a maximum flow on the network is $O(n^{2/3})$. Suppose that k blocking flows have already been computed. Then the distance from the source to the sink in the residual network is at least k, so the residual maximum flow is at most $\frac{4n^2}{k^2}$. Since each augmenting path saturates at least one arc, and the arcs have unit capacity, the number of remaining paths is at most $\frac{4n^2}{k^2}$. Since each blocking

flow finds at least one path, this is also a bound on the number of blocking flows remaining. So the total number of blocking flows is

$$k + \frac{4n^2}{k^2}$$

which, if we let $k = n^{2/3}$, equals

$$n^{2/3} + \frac{4n^2}{n^{4/3}} = n^{2/3} + 4n^{2/3} = O\left(n^{2/3}\right)$$

Problem 2:

a) Suppose we wish to match s students to r recitations with $k = 1$ students per recitation. This is simply a bipartite matching algorithm. We define a graph $G = (V, E)$ with one vertex corresponding to each student and each recitation, and connect each student's vertex to the vertex for each recitation they can attend. We create a source vertex s and connect it to each student, and a sink vertex t connected to each recitation. All edges have unit capacity. As proven in the class notes, we can compute a maximum flow over this bipartite graph using blocking flows in $O(m\sqrt{n})$ time. This corresponds to an optimal assignment; a particular student is assigned to a particular recitation if the edge between the corresponding vertices in the graph has flow 1.

b) For arbitrary k, we use the same graph as before. However, since k students are now allowed to be matched to each recitation, we create k vertices for each recitation, connected to the sink with unit capacity edges. We connect each vertex corresponding to a student to *every* vertex corresponding to each recitation that the student can attend. Again, a maximum flow corresponds to an optimal assignment. Note that k students can be assigned to each recitation, since there are k vertices for the recitation (each one corresponding to one "slot" for a student in the recitation), but each student can still only be assigned to one recitation, because there is still only one vertex per student and it is connected to the source by an edge of unit capacity. Thus, finding a maximum flow finds a correct optimal assignment.

c) Note that the network created above is still a unit network, since we have merely added more vertices and edges; the capacity of each edge is still 1. So the network is still a bipartite graph with unit capacity. As before, we can still find a maximum flow in $O(m\sqrt{n})$ time using blocking flows.

fine because blows up by $k^{3/2}$ bc of

Problem 3:

Let $D = \{d_{ij}\}$ be a p by q table of non-negative integers, and r_i be the sum of the elements in the ith row and c_j be the sum of the elements in the jth, and suppose the row and column sums r_i and c_j are disclosed along with a set of matrix elements Y. We describe a flow-based algorithm for identifying whether the value of any suppressed elements can be exactly determined, and if so the associated value.

We begin by translating the problem into a flow network. Create a vertex (call it r_i) for each row i, and connect it to the source s with an edge of capacity r_i minus the sum of the known elements in Y that are in row i. Similarly, create a vertex c_j for each row j, and connect it to the sink t with an edge of capacity c_j minus the sum of the known elements in Y that are in row j. Next, for each entry d_ij of D that is not in Y, we create an edge between vertex r_i and c_j with infinite capacity.

To show the correctness, we first claim that a maximum flow on this network corresponds to a valid assignment of values to the suppressed elements of D. To read the elements of d from the flow, define the value of the element d_ij to be the flow through the edge connecting vertices r_i and c_j.

We now show that the flow thus defined obeys the row and column sum constraints. First observe that in any maximum flow, the edges between s and r_i and the edges between c_j and t will be saturated. This follows from the fact that the edges between the r_i and c_j are derived from a matrix D that satisfies the row and column sums. The total amount of flow is $\sum_i r_i = \sum_j r_j = \sum_{i,j} d_{ij}$ minus the elements in Y. The flow through r_i is the sum of the flow of the edges that connect to r_i, which represent the values in row i. But the capacity of the flow from the source to r_i is r_i minus the sum of the disclosed elements in that row. If we also add in the disclosed elements in r_i, we find that the sum of our values for d_{ij} satisfies $\sum_j d_{ij} = r_i$, as desired. A symmetric argument shows the same result for the columns c_j, i.e. that the values we have found for d_{ij} satisfy $\sum_i d_{ij} = c_j$. So we have shown that a maximum flow in this network corresponds to a feasible assignment of the d_{ij}.

Of course, this cannot find a unique assignment for all d_{ij}, since for protected, suppressed values there can be no such assignment. We have only found the value of an element if it is unprotected, and an element is unprotected only if the corresponding edge has the same value in *every* maximum flow.

Lemma 1. *Any maximum flow can be generated from any other maximum flow by augmenting along a cycle.*

Proof. Recall that any flow can be decomposed into a set of s → t paths and a set of cycles. Two maximum flows cannot differ by simply adding or removing a s → t path, as this would change the value of the flow, and one of the flows would thus not be maximum. However, augmenting along a cycle does not change the value of the flow. Thus, we can generate new maximum flows by augmenting along cycles. This condition is also sufficient. □

Corollary 2. *If an edge is not contained in a cycle in the residual network in a maximum flow, it will have the same flow value in every maximum flow.*

Proof. The edge is not contained in a cycle, so augmenting along cycles to generate new maximum flows will never change the flow through that edge. □

Thus, we see that an unprotected element corresponds to an edge that is not contained in any cycles in the residual network. This gives us an algorithm for computing the unprotected elements and their values: we first compute the maximum flow over the network, then generate the residual network and use breadth-first search to identify cycles. Any $r_i \to c_j$ edge whose reverse edge is not contained within any cycles in the residual network corresponds to an unprotected element, and its value is given by the flow through that edge. Depending on the max-flow algorithm used, the algorithm can run in $O(n^2 m) = O(n^4)$ for $n = O(\max p, q)$.

Problem 4:

a) To solve the minimum flow problem on a graph G, we first use a maximum flow algorithm (with zero lower bounds) to identify a feasible flow if one exists, then apply a maximum flow algorithm again to reduce it to the minimum flow.

We begin by identifying a feasible flow. To do so, we first construct a graph G' identical to G, but we define the capacity of an edge to be the difference between the upper and lower bounds: $c_{ij} = u_{ij} - l_{ij}$. Then, for each edge ij with non-zero lower bound l_{ij}, we create an edge between s and j and another between i and t, with capacity l_{ij}. This corresponds to "removing" l_{ij} units of flow at i and "replacing" them at j, since the flow is forced; the capacity of ij is decreased by l_{ij} since we are treating this "forced" flow separately. We then compute a maximum flow over G'. If all of the edges we added are saturated in the maximum flow, then there exists a feasible flow; if not, then there is some edge with a lower bound that cannot be satisfied by any flow. To extract the feasible flow on G from G', we simply remove the extra edges we added, and add the lower

bound l_{ij} to each flow f_{ij}. The result is a feasible flow on G, since it clearly satisfies the lower and upper bounds, and conservation is satisfied because our edges between s and j, and between i and t ensured that l_{ij} flow can be pushed through the edge.

We now show how to convert a feasible flow f on a graph G into a minimum flow on G. For graphs with lower bounds, we define the residual network differently: let G_f be the residual network with flow f, and define the residual capacity of each edge ij in G_f to be $u_{ij} - f_{ij} - l_{ji}$. This reflects the fact that we can increase the flow f_{ij} up to the capacity u_{ij} as before, but we can no longer violate the lower bound constraint. Note that, as before, if the residual capacity of an edge in the residual network is greater than zero, it is possible to augment along that edge without violating either the lower or upper bound constraints. Thus, we can apply an augmenting-path algorithm (e.g. blocking flows, etc.) on the residual network, since these do not depend on the details of the definition of the residual capacity. However, since we wish to *decrease* the flow from s to t, we compute augmenting paths from t to s, i.e. treating t as the source and s as the sink. Adding an augmenting path to the flow creates another feasible flow, since increasing the capacity on each edge does not violate either the lower or upper bound constraints. We need only show that the flow produced is a minimum flow. When the residual network contains no augmenting paths, the value of the flow is equal to the capacity of some cut. By our definition of the residual capacity of an edge, and since the augmenting paths are being computed from t to s, the capacity of the cut is thus $\sum_{(i,j)\in S\times T}(l_{ij} - u_{ji})$, the lower bound on the capacity of the cut. Thus, the correctness of this algorithm is proven by the minimum-flow – maximum-cut theorem shown below.

b) Let f be a minimum flow on a network G. We show that $|f|$ equals the maximum lower bound on cut capacity of all $s-t$ cuts. As above, define the residual network G_f that has residual capacity $u_{ij} - f_{ij} - l_{ji}$ for edge ij. Define the lower bound on the capacity of a $s-t$ cut to be $\sum_{(i,j)\in S\times T} l_{ij} - \sum_{(i,j)\in T\times S} u_{ij} = \sum_{(i,j)\in S\times T}(l_{ij} - u_{ji})$.

We first show that for any cut S, T, the minimum flow must be at least as large as the lower bound on the cut capacity. At every edge the flow must be greater than the lower bound minus the upper bound in the opposite direction, so we have $f_{ij} \geq l_{ij} - u_{ji}$ for all $i \in S, j \in T$. !Since any flow can be decomposed into a set of $s-t$ paths and cycles, and every $s-t$ path crosses the cut in the forward direction once more than it crosses the cut in the reverse direction, the entire flow passes through the cut. Summing over the cut, $|f| \geq \sum_{(i,j)\in S\times T}(l_{ij} - u_{ji})$.

We next show that there exists a maximum lower bound on cut capacity that equals the minimum flow. Note that (since negative flows are allowed when they satisfy the lower and bound constraints), a minimum flow from s to t is equivalent to a maximum flow from t to s. So we observe that if a flow f is minimum, there exists no augmenting path from t to s in the residual network. Thus, the residual network G_f is disconnected; there exists a set S containing s and a set T containing t, and these sets form a cut. Since these sets are disconnected in the residual network, for all $i \in S$ and $j \in T$, $c_{ji} = 0$. So in f, $u_{ji} - f_{ji} - l_{ij} = 0$, or equivalently (applying skew-symmetry) $f_{ij} = l_{ij} - u_{ji}$. Thus, $|f| = \sum_{(i,j)\in S\times T}(l_{ij} - u_{ji})$, i.e. $|f|$ is equal to the lower bound on cut capacity of the cut S, T.

c) We convert this problem to a flow network. Create a vertex $\mathsf{a_i}$ to represent the start of lecture i, and $\mathsf{b_i}$ to represent the end of lecture i. Connect these two vertices with an edge with lower bound 1; this represents the requirement that at least one student attend one lecture. Then connect vertex $\mathsf{b_i}$ to $\mathsf{a_j}$ if $a_j - b_i > r_{ij}$, i.e. if the beginning of lecture j can be reached from the end of lecture i while satisfying the travel time requirements. Then connect the $\mathsf{a_i}$ vertices to a source s, and the $\mathsf{b_j}$ vertices to a sink t. All edges have infinite capacity and no lower bound, except as previously stated. Then computing the minimum flow gives an assignment of students to lectures. Each unit of flow from the source to the sink gives a "path" that one student should follow, traveling to and attending lectures as appropriate. The lower bound constraint ensures that every lecture will be attended by at least one student, and the minimality of the flow shows that this assignment requires the least number of students possible.