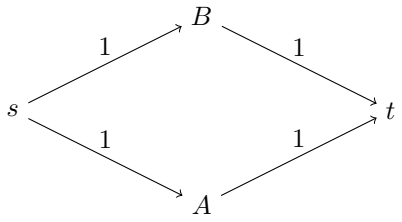
6.854 Advanced Algorithms

Problem Set 5

John Wang
Collaborators: Jason Hoch

Problem 1-a: An edge is upward critical if replacing its capacity c by any $c' > c$ increases the maximum flow value. Does every network have an upward-critical edge? Give an algorithm to identify all upward-critical edges in a network. Its running time should be substantially better than that of solving m maximum-flow problems.

Solution: Not every network has an upward-critical edge. Consider the graph below:



We see that the capacities on all of the edges is 1. However, increasing the capacity of any edge will not increase the flow, since the max flow will always stay at 2 (one unit of flow coming from the s, A, t path and the other coming from the s, B, t path. Therefore, we see that not all networks have an upward-critical edge.

To find an algorithm for detecting upward critical edges, we will first note that if we increase the capacity of an edge, and afterwards, the residual graph of a max flow now has an augmenting path from s to t, then the max-flow can be increased. Assuming integer capacities, increasing the capacity by 1 suffices for creating a new augmenting path. Therefore, we can create an algorithm to find these new augmenting paths.

First, we perform a BFS to determine the connected components in the residual graph of a max flow. This is done by starting at the source s and finding all nodes that s can navigate to (assuming that 0 weight edges are non-existent). Then, we will go to t and reverse all of the edges in the residual graph. We will perform a BFS to find all the backward connected components of t. This backward BFS will find all nodes v which have a path to t. Now, we can label the first set of components S and the second set T, and it is sufficient to find an edge that connects S to T when its capacity is increased.

In order to find such an edge, we can iterate over all edges $e = (u, v)$ in the residual graph, and see if $u \in S$ and $v \in T$ or vice versa. If this is the case, then we can increase the edge's capacity and create an augmenting path from s to t, thus increasing the max flow. Note that all such possible augmenting paths will be found by the algorithm.

The total time of this algorithm requires $O(m+n)$ for the two BFS searches for the connected components and $O(m)$ for iterating over the edges and checking if the edges are in sets S and T. Thus, the total time of the algorithm is $O(m + n) = O(m)$. □

Problem 1-b: An edge is downward critical if replacing its capacity c by any $c' < c$ decreases the maximum flow value. Is the set of upward-critical edges the same as the set of downward critical-edges? Describe an algorithm for identifying all downward critical edges, and analyze your algorithm's worst-case complexity. Its running time should be substantially better than that of solving m maximum-flows.

Solution: The set of downward-critical edges is not the same as the set of upward-critical edges. Consider the figure that was presented in part a, with all edges of capacity 1. If we decrease the capacity of any of the edges by 1, we will decrease the maximum flow by 1 as well, since we will remove the flow path of either s, A, t or s, B, t with such a decrease in capacity. This means that every edge is in the set of downward-critical edges. Since we already showed that the set of upward-critical edges for this graph is empty, we see that these two sets are not always the same.

Now notice that if any edge $e = (u, v)$ can be removed and the maximum flow does not change, then that edge e is not downward-critical. If the maximum flow does change, then e must be downward critical because

we can removed $c' = w(e)$ from the current capacity $c = w(e)$ of the edge and decrease the maximum flow value. Thus, we need to find all edges, which when removed, no longer have the sames flows from s to t. We can do this by finding looking at each edge $e = (u, v)$ and finding out if u and v are connected in the residual graph with a path of flow equal to the flow that used to go over $e = (u, v)$.

We can use BFS to examine each edge $e = (u, v)$ and examine whether there is a path from u to v in the residual graph after removing e whose flow is greater than or equal to the flow over e. If there does not exist such a path, report e as downwards critical. This requires m iterations of BFS, so the total runtime is $O(m(m + n)) = O(m^2)$. $\square$

6.854 Advanced Algorithms

Problem Set 5

John Wang
Collaborators: Jason Hoch

Problem 2-a: Show how to solve the minimum flow problem by using two applications of any maximum flow algorithm that applies to problems with zero lower bounds on edge flows (e.g. the algorithms described in lecture). Your algorithm should detect if there is a feasible flow and, if there is one, return a minimum flow.

Solution: □

6.854 Advanced Algorithms

Problem Set 5

John Wang
Collaborators: Jason Hoch

Problem 3-a: Suppose that the numbers of faculty and students are equal, each student wants to meet exactly d faculty, and each faculty member is on the request list of d students. Conclude that one can schedule a single slot in which every student is meeting someone.

Solution: We will construct a bipartite graph where the left side of the graph is composed of the n students, and the right side is composed of the n faculty members. An edge connecting a student u to a faculty member v in the graph implies that the student wants to meet faculty member v. This implies that there are exactly d outgoing edges from any u in the set of students S and exactly d incoming edges to any v in the set of faculty members F.

We can construct a flow graph by creating a source s and a sink t. From the source, there are edges (s, u) for all $u \in S$ with capacity 1. For the sink, there are edges (v, t) for all $t \in F$ with capacity 1. For the edges (u, v) where $u \in S$ and $v \in F$, we will set the capacity to 1. To show that we can schedule a single slot in which every student is meeting someone, we need to find a perfect matching between S and F.

In order to do that, consider the following flow:

$$f(u,v) = \begin{cases} 1 & \text{if } u = s \\ 1/d & \text{if } u \in S, v \in F \\ 1 & \text{if } v = t \end{cases} \tag{1}$$

We see that this is a valid flow through the graph since at each node $u \in S$, there are exactly d outgoing edges of flow $1/d$, and a single incoming edge of flow 1. For each node $v \in S$, there are d incoming edges of flow $1/d$ and a single outgoing edge of flow 1. Moreover, we see that all of these flows are within their capacity limits by inspection. This ensures that this is a valid flow. We see that the value of this flow is n, so that its cut is of capacity n as well.

By the integrality theorem, there exists some max flow for which all flow values are integers. This implies that there exists some max flow, of value greater than or equal to n, which has a min-cut of capacity n. This means that there must be n edges crossing the min-cut (since all edges have capacity at most 1), which further implies that there exists n paths from s to t of capacity 1. This shows that each student can be matched up to a faculty member. □

Problem 3-b: Conclude that it is possible to schedule all the meetings to take place in d time slots.

Solution: We will prove that the capacitated graph we constructed above has d perfect matchings where no matching has an edge in common (d different max flows without shared edges in the min-cut). If this is the case, then it is sufficient to set these perfect matchings in arbitrary order in order to fill up all of the meetings in d time slots. We will call a graph where each student has exactly d faculty requests, and each faculty member has exactly d students on their request list, a d-request graph.

We will prove the following lemma: if G is a d-request graph, then G has d perfect matchings, no pair of which have any edge in common. We show this by induction on d. For $d = 1$, we see that a perfect matching is trivially satisfied by following all of the edges, so that each fauclty member is paired with the only student on their list. Now suppose $d = k$ and we have proven this fact for all d through $k - 1$. Now let there be subsets $A \subset S$ and $B \subset F$.

Let us define $N_G(A) = \{u \in V(G) | u \text{ is adjacent to v} \in A\}$ as the neighborhood of A. Now suppose by contradiction that $|N_G(A)| < |A|$. We know that $\sum_{u \in A} \sum_{v \in F} e(u, v) = d|A|$, by the fact that A is part of a d-request graph. This implies by the pigeonhole principle that there exists a vertex in $N_G(A)$ whose degree is greater than d. This contradicts the fact that G is a d-request graph (and that there are exactly d requests for all nodes in S). Thus, we find that $|N_G(A)| \geq |A|$.

Since we know that $|N_G(A)| \geq |S|$ for all $S \in A$, Hall's Marriage Theorem (Theorem 26.3-4 in CLRS) shows that G has a perfect matching. Now we can remove every edge e which was in this perfect matching, and create a new $k - 1$-request graph. This is true because each edge removes 1 from the incoming edges

for each node $v \in F$ and the outgoing edges for each node $u \in S$. We know that the $k-1$-request graph has $k-1$ perfect matchings by our inductive hypothesis. This implies that there are $k-1+1=k$ perfect matchings for a k request graph. This completes the proof. □

Problem 3-c: Consider an arbitrary set of desired meetings. Obviously one needs at least as many slots as there are faculty to meet a given student, and students to meet a given faculty. Prove that one can arrange all meetings with no more slots than this number s.

Solution: Let $s = \max\{d_f, d_s\}$ where d_f is maximum number of faculty on the list of any student and d_s is the maximum number of students who any faculty is required to meet. □

Problem 3-d: Show that the schedule can be computed in $O(s^2n^{3/2})$ time, where s is defined in (c) and n is the total numbmer of students and faculty members.

Solution: Notice that our bipartite graph has n vertices and $m = sn$ edges. This is because there are at most s outgoing edges on any vertex $u \subset S$ and s incoming edges on any vertex $v \subset F$. We have also shown in class that maximum flow can be achieved in a bipartite graph in $O(m\sqrt{n}) = O(sn^{3/2})$ by using a reduction to a unit graph and performing blocking flows on the unit graph.

We have already shown in part (c) that we can arrange all meetings with no more than s time slots. This implies that we can find s maximum flows iteratively. On the ith iteration, we will find a max flow of the graph G_i, schedule student-faculty member meetings according to the matchings provided by the max-flow (edges leading from students to faculty member nodes will each be a meeting), then remove all the (u, v) edges where $u \in S$ and $v \in F$ from the max-flow. This will leave a graph G_{i-1} with all previous max-flow matchings removed. Performing s iterations of this will result in a complete schedule over s time slots.

The algorithm requires $O(sn^{3/2})$ work during each iteration. Since there are s iterations, the total cost is $O(s^2n^{3/2})$, which is what we wanted. □