Homework 5 (100 points)

Due in class on 4th December, 2017

There are 5 questions in this homework.

1) (10 points) For a bipartite graph $G(A \cup B, E)$, let M' be a maximum cardinality matching and let M be any other matching. Recollect that the graph $G'(A \cup B, M \oplus M')$ consists only of cycles, paths and isolated vertices. Prove that this G' contains exactly $|M'| - |M|$ paths of odd length.

2) (8+7+5 points) Let M' be the maximum cardinality matching for a bipartite graph $G(A \cup B, E)$.

   a. Let M be a matching at the end of the first iteration of Hopcroft-Karp Algorithm. Prove that $|M| \geq \frac{|M'|}{2}$,
   b. For some integer $k \geq 0$, let M_k be the matching after finishing k iterations of Hopcroft-Karp Algorithm. Prove that $|M_k| \geq \left(1 - \frac{2}{k}\right)|M'|$
   c. For $0 < \epsilon \leq 1$ an ϵ-approximate matching is any matching M with the property that $|M| \geq (1 - \epsilon)|M'|$. Use part (b) to show that an ϵ-approximate matching can be computed in $O\left(\frac{m}{\epsilon}\right)$ time.

3) (20 points) Give a polynomial time algorithm for the following minimization analogue of the max-flow problem. You are given a directed graph G(V,E), with a source s and a destination t and capacities l_e for each edge $e \in E$. We define a flow f, and a value of a flow, as usual, requiring that all nodes except s and t satisfy the conservation condition. However, the given numbers are lower bounds on the edge flow, i.e., they require that $f_e \geq l_e$, for each edge e and there is no upper bound on the flow value of the edges.
   a. Give a polynomial time algorithm that finds a feasible flow of minimum possible value. (Hint: Start with any flow from s to t which obeys the lower bounds, and try to send a maximum flow from t to s in a suitable modification of the graph G)
   b. Prove the analogue of max-flow min-cut theorem for this problem.

4) (15+ 10+5) In the Hungarian Algorithm described in the class, in each iteration, we start with a feasible matching M and a set of labels (dual weights) $\ell(v)$ for every $v \in A \cup B$. In each iteration we apply the following steps (here A_F and B_F denote the free vertices of A and B).
   a) We add a dummy vertex s to the residual graph and add an edge from s to every free vertex in B_F. Each such edge is given a cost 0. Every other edge in the residual graph is given a cost equal to the slack (with respect to the current labels) of that edge.

b) We compute the shortest path from s to every node in the graph. Let L_u denote the shortest path cost from s to any node u.

c) We find the free vertex a' in A_F that has the smallest shortest path value L, i.e., $L = \min_{a \in A_F} L_a$ and $a' = \arg\min_{a \in A_F} L_a$. Let P' be the shortest path from s to a' in the residual graph and let P be the corresponding augmenting path.

d) Update the dual weights as follows. For any vertex $v \in A \cup B$, if $v \in A$ and $L_v \leq L$, then $\ell(v) \leftarrow \ell(v) - (L - L_v)$ and if $v \in B$ and $L_v \leq L$, then $\ell(v) \leftarrow \ell(v) + (L - L_v)$.

e) $M \leftarrow M \oplus P$ (i.e., augment M along P)

Answer the following questions.

(i) (15 points) Show that after step (d), the new dual weights and the matching is a feasible matching

(ii) (10 points) Show that after step (d), every edge on the augmenting path P has zero slack.

(iii) (5 points) Show that after step e) the new matching and the dual weights are also feasible.

5) (10 points+10 points) The following theorem holds for any planar graph: There is a set of $O(\sqrt{n})$ vertices (called the separator set) whose removal from a n-vertex planar graph partitions the graph into disjoint subgraphs each having at most 2n/3 vertices. Given a n-vertex planar graph, assume that such a separator set can be computed in O(n) time.

a. Given an unweighted planar bipartite graph, use the planar separator theorem to design a divide-and-conquer algorithm for computing the maximum cardinality matching. This algorithm should take $O(n^{1.5} \log n)$ time.

b. Give a divide and conquer algorithm to compute minimum-cost matching in a weighted planar bipartite graph. The execution time of this algorithm should be $O(n^{1.5} \log n)$ (You can assume that this graph has a perfect matching and that all the edge costs are positive; please DO NOT design a scaling algorithm)

Theory of Algorithms. Homework 5

Peter Racioppo

1.)

Proof by construction:

All of the vertices incident on the edges that belong to M must also be in some M', since otherwise we could draw an edge to this vertex. Call the edges that are in both M' and M the set X. Imagine starting with the edges in M'-X and adding in the edges in M-X in any order, so that we start with |M'-X| paths of length one. Adding in a path from M results in a new set of paths of cardinality one less than the cardinality of the set of paths in the previous step, because we're concatenating two of the old paths. We can repeat this process exactly |M|-X times. Thus, we end up with a set of paths of cardinality |M'-X| - |M-X| = |M'-M|. The paths must always be odd because they are alternating paths beginning and ending in edges of M'.

Neater proof:

Suppose that there are d > |M'|-|M| augmenting paths for M. Then M can be augmented until it has greater cardinality than M'. Contradiction.

Suppose that there are d < |M'|-|M| augmenting paths for M. Define W as the augmentation of the d paths with M. The cardinality is then less than |M'|. But W is a maximal matching, since there are augmenting paths (Berge's Theorem). Contradiction.

So we've bounded the number of augmenting paths from below and above as |M'|-|M|.

2.)

a.) Each matched edge in M can at most be replaced by two matched edges in M'. (If an edge in M is not in M', the worst case is that both of the vertices incident on this edge are incident on other edges in M'.)

b.) By (1), there are at least |M'|-|Mk| augmenting paths of odd length in G', the disjoint union of M and M'. Each such path is at least of length k. Also, there is always one more edge from M' than from Mk in each of these, so more than half of the k edges from each of the |M'|-|Mk| paths belongs to M'. Thus, (|M'|-|Mk|)*(k/2) <= |M'|.

c.) At the kth step, the algorithm has performed k iterations, each of which takes O(m) time, so the run time until then is O(mk). Define k = ceil(2/epsilon). Then the inequality holds by (b) and the algorithm is O(m/epsilon).

3.)

a.) Use DFS/BFS to find a directed s-t path and push a flow equal to the greatest bound on this path. Keep doing this until all edges have met their minimum capacity. If an edge is part of multiple such paths, add all the values from the various iterations; this ensures flow conservation. The full operation takes at most O(E), where E is the number of edges, so this part is polynomial. Define the edges in the residual graph as having weights equal to the flow currently pushed minus the capacity: la - le. Perform Ford-Fulkerson on the residual graph.

Pushing down the paths in either direction is conservative, by construction. Pushing from s to t is viable, by construction. Pushing from t to s is also always viable since we can push no more than an excess flow through a residual edge, by construction. Ford-Fulkerson guarantees a maximum flow in the residual graph; the correctness of the algorithm follows.

b.) For any cut (A,B) with a subset of vertices A, v(f) = f_out_A - f_in_A.

The flow across an edge must be greater than or equal to its constraint capacity. Summing across any cut, the flow across the cut must be greater than or equal to the sum of the constraint capacities. (actually, that's not true.) In particular, this must hold for the maximum cut. (ok, suppose that's the case.)

After the final iteration of Ford-Fulkerson, consider the set A, the set of vertices reachable in the residual of the residual graph from t, and Ac, the set of remaining vertices. Now, consider the cut of the edges between A and Ac. By construction, the edges in the double residual graph must be directed from Ac to A, or Ac would be reachable from t. This requires that the edges in the original residual graph going from A to A must be saturated and those going from Ac to A must have flow zero. Thus, in the original residual graph, the capacity of this cut is equal to the flow. Thus, in the original graph, there exists a cut where the slacks are zero at every edge. In other words, there exists a cut in the original graph which has a constraint capacity exactly equal to the flow. This must be the minimum flow, since every flow must be greater than or equal to the max cut. In summary, max cut = min flow.

4.)

i.) There are four cases:

* If Lb > L and La > L, there are no updates to the vertex labels.

* If Lb <= L and La <= L, l'(b)+l'(a) = l(b)+l(a) - L + L + (Lb-La)

But Lb - La <= ce - (lb+la) (since this is in the residual graph)

so l'(b)+l'(a) <= l(b)+l(a) +ce - (lb+la) = ce

* If Lb <= L and La > L, l'(b)+l'(a) = l(b)+l(a) - (L-Lb) <= l(b)+l(a).

* If Lb <= L < La by construction. => L - Lb < La- Lb = ce, provided that a is on the shortest path to b. (is this true?)

l'(b)+l'(a) = l(b)+l(a) + (L-Lb) <= l(b)+l(a) + ce.

ii.) Every vertex on the shortest path P must satisfy Lv <= L and as such the vertex labels are unchanged. But all of the corresponding edges have to be in the equality graph, so they already have slack zero. The only edge to worry about is the one that terminates in a'. Now the second vertex on this edge, namely a', is unchanged, but the first one falls under the case Lb <= L <= La, so l(b)' =l(b)+(L-Lb) = l(b)+ce. But we're in the residual graph, so ce is the slack. We've added in the slack to this edge, so now it's tight.

iii.) All elements of M and P are in the feasibility graph. So the disjoint union of the two also only includes feasible elements.

5.)

a.) The divide step has already been described. Assuming that separating the graph approximately divides it into two equal pieces, each piece is of size O((n-sqrt(n))/2) = O(n/2). In the worst case, the bigger set has (2/3)*n elements. In the worst case, the bigger set in the (k+1)th step always occurs as a subset of the bigger set in the kth step, which would imply that the number of steps m before we get to sets of cardinality 1 is given by: n*(2/3)^m = 1 => m = logn, with base 3/2. At the kth level, we have to make a separator set in time (n*(2/3)^k) = (n*(2/3)^logn). Summing from k=1 to logn, the total time is still O(n).

The two separated sets A and B at each iteration have no shared edges. Use Ford-Fulkerson to match vertices in the union of the separated and separator sets A U S and B U S. The recursion is 2*T(2n/3). The run time at each of the logn iterations is O(m) where m is the number of edges, and this is O(n), since the graph is planar. There are n*sqrt(n) edges in the first step, so (n^1.5)*logn bounds the total run time.

For the merge step, we must consider A U S U B. There are generally some double matchings in S. For each double matching, delete one edge randomly. We can be no more than sqrt(n) matched edges away from the maximal matching. Run the Hungarian method on the sqrt(n) vertices. It's O(nm) and there are no more than O(n) edges since the graph is planar, so the run time of this step is O(n*sqrt(n)).

b.) The divide step is the same as in (a).

Hopcroft-Karp takes O(m*sqrt(n)) time. Since this is a planar graph, m = O(n), so do H-K in O(n*sqrt(n)).

For the merge step, we could finish the matching using the Hungarian method in O(nm) time. The number of edges between the separated and separator sets is O(n+sqrt(n)) = O(n), since the graph is planar, so the total time for each of the logn iterations would be O(n*sqrt(n)).