CS 6150: HW4 – Graphs, Flows and Cuts

HINTS/SOLUTIONS

1. (a) (2 points) Let G be a weighted, undirected graph with all edge weights being distinct integers. Prove that there is a *unique* minimum weight spanning tree.

   Suppose, if possible, S, T are two distinct MST's. Let the edge weights be $s_1 \leq s_2 \leq \ldots, s_{n-1}$ and $t_1 \leq t_2 \leq \cdots \leq t_{n-1}$. First, note that $s_1 = t_1$, because otherwise, if say $s_1 < t_1$, then adding the (unique) edge of weight s_1 to T will form a cycle, and removing one of the cycle edges will result in an MST of weight strictly smaller than that of T – a contradiction.
   We can show by induction that $s_k = t_k$ for all k. Suppose this is true until $k-1$. Now, we can use a reasoning similar to the above: say $s_k < t_k$, and consider the cycle in T formed by adding the edge of weight s_k. This cycle must have an edge of weight $\geq t_k$, because the smaller edges are all also in S, and S is acyclic (it's a tree). Removing this edge and adding s_k again gives a contradiction.
   Thus we have $s_k = t_k$ for all k, and thus $S = T$.

   (b) (3 points) Even if the weights are not distinct, there could be graphs in which the MST is unique. Given a weighted undirected graph G, give a polynomial time algorithm to determine if G has a unique minimum weight spanning tree.

   First find an MST in G. Then remove each edge of it and compute the MST of the remaining graph. If the graph has two different MST's, they differ in at least one edge, and thus this process will find the other MST. If the process fails, it means that the MST is unique.

2. (a) (2 points) Let G be a directed graph with all edges having an integer capacity. Prove that for any s, t, there exists a maximum flow between s, t in which every edge has an integral flow on it. [This is what we used in class to reduce the maximum matching problem in bipartite graphs to flows.]

   Consider the augmenting path algorithm. The amount of flow we send in each step is the residual capacity on some edge. This is integral, because the residual graph always has integer capacities. This implies that all the flow values we get are integral.

   (b) (4 points) Suppose you have already computed the max $s-t$ flow in a network with *integer* capacities. Give an efficient algorithm to update the maximum flow after one edge's capacity is increased/decreased by 1 (i.e., give algorithms for both the cases). The running times should be significantly faster than recomputing the maximum flow from scratch.

   Let n, m be the number of vertices and edges in the graph resp. Assume we have the flow values f_e fo every edge computed. From part (a), we may assume they are integral. (I don't think there's an easy way to solve this problem if you do not make this assumption.)
   If we increase the capacity of some e by 1, the max flow can only change by 1 (because any cut can increase by at most 1). Thus, in the residual, we can simply look for an augmenting path, and modify the flow if one is found; the update time is thus $O(m+n)$.
   Now suppose the capacity $C(uv)$ of an edge uv is decreased by 1. If $f_{uv} < C(uv)$, there is nothing to do – the same flow is feasible. Thus suppose $f_{uv} = C(uv)$. We will first move to a 'feasible' flow with the new capacities, and then try to find an augmenting path. So

the first step is to find an $s-t$ path in the current flow that goes through uv, reduce the flow along that path by 1. This can be done in two steps: first find a path from u to s in the 'reversed' flow graph (the flow graph is simply the weighted directed graph in which the edge weight is the current flow value) via a BFS, and then find a path from v to t in the flow graph. These two paths cannot intersect, because the flow graph is acyclic (! this is a nice property that's true of all the flow algorithms we've seen). This yields an $s-t$ path going via uv, on which we reduce the flow by 1. This yields a feasible flow for the new capacities, and we simply find an augmenting path if possible.
Overall, we perform 3 graph searches, so the run time is $O(m+n)$.

(c) (4 points) Let G be a directed graph with all edges having capacity 1. Further, suppose the *in-degree* of every vertex is equal to its *out-degree*. Prove that for any two vertices s, t and integer $k \geq 1$, there exist k edge disjoint paths from s to t iff there exist k edge disjoint paths from t to s.

The trick is to observe that the in-degree = out-degree property is also true for *sets* of vertices. Take any subset S of vertices. We claim that the total number of edges entering S is equal to the toal number of edges leaving S. There are many ways of seeing this. A proof by induction is easy.. (detail omitted here)
Once we observe this, it means that the min $s-t$ cut has the same value as the min $t-s$ cut. This implies the desired result (via the max-flow min-cut theorem).

3. We will see some more examples of problems reducing to flow/matching.

(a) (5 points) Consider the problem of forming a univeristy wide faculty committee that has one representative from each department. Suppose the university has n departments, and m faculty. Each faculty member may have appointments at multiple departments, and thus he/she may be a represetative for any of those departments. However, two departments are not allowed to pick the same faculty member as their representative.

Note that forming such a committee easily reduces to a max flow problem. Now, suppose we have the additional constraint that the committee has equal representation from all ranks of professors (assistant, associate, full). Give a polynomial time algorithm that incorporates this constraint and comes up with a committee, or concludes that it is impossible.

Let us assume that n is divisible by 3, because otherwise, it is clearly impossible to achieve the same number of representatives of each rank.
We construct a 3-layer graph, where the layers (in order) correspond to departments, faculty, rank. Thus the layers have $n, m, 3$ nodes respectively. Additionally, there's a source s, connected to each node in the first layer, with an edge of capacity 1. So also, there's a sink t, and there are edges from each of the three nodes in the final layer to t, each of capacity $n/3$. There's a unit-capacity edge between dept j and professor i if i can act as a representative for dept j. So also, if faculty i is of rank r, there's a unit capacity edge from i to r.
The valid committee can be formed iff there is a flow in this graph of value n. In this case, finding an integer flow, and 'reading off' all the professors i with a non-zero incoming flow would give a valid committee.

(b) (5 points) Suppose you are given an $n \times n$ checkerboard with some of the squares deleted. You have a collection of 2×1 dominoes, which can be used to cover any two adjacent squares. Describe an algorithm to determine if the board can be "tiled" with dominoes, i.e., if we can place dominoes on the board (either horizontally or vertically) such that each one covers two squares (neither of which is deleted), and no two dominoes overlap. As an example, consider Figure 1.

Assume that the squares in the board are colored black and white in the usual way. Construct a bipartite graph, with the non-deleted white and black squares on either side. Place an edge iff the two squares are adjacent on the board (i.e., if it's allowed to place a domino covering the two squares).

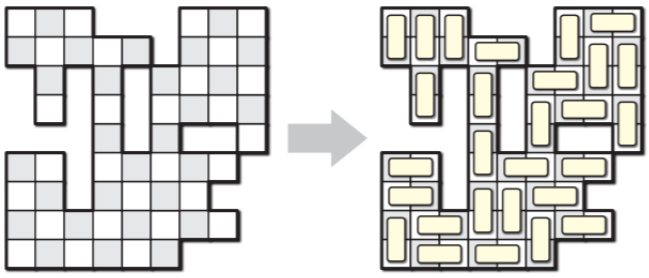

Figure 1: Checkerboard with some squared deleted on the left, and a feasible domino tiling on the right.

There's a valid tiling iff there is a perfect matching in the graph above. Thus, we have a poly time algorithm. (Construct a flow network with extra source, sink, unit-weight edges pointing right, and look for a flow of value equal to number of nodes/2.)

(c) (5 points) A *cycle cover* of a directed graph $G = (V, E)$ is a set of vertex-disjoint cycles in the graph that cover all the vertices. Design a polynomial time algorithm for finding such a cycle cover, or conclude that it is impossible.

Let $G = (V, E)$ be the given graph. Now, form a bipartite graph with a copy of V on either side, called S, T, resp. If $(u, v) \in E$, then we place an edge between the copy of u in S and the copy of v in T. A perfect matching can be used to obtain a *chain* in G, as follows: start with any $u \in V$, and view the matching as giving the 'next' vertex in the chain. Since the number of vertices is finite, we must reach a vertex that has already been reached (call this x). If $x \neq u$, then it means that two distinct vertices map to x – a contradiction. Thus, $x = u$, and we have a cycle. Now start with a vertex not already covered, and perform the same procedure until all vertices are exhausted. This yields a cycle cover.

(d) (5 points) Alice and Bob play the following game: Alice starts by naming an *actress* A_1. Bob must then name an *actor* B_1 who has co-appeared in a movie with A_1. Then Alice must name an actress A_2 who co-apparead with B_1, and so on. (Alice must always pick from the set of actresses, and Bob must pick from the set of actors.) The catch is that the players are not allowed to name anyone they have named already. The game ends and a player loses if he/she cannot name an actor/actress who hasn't been named already.

Suppose we are given as input a set of all "eligible" movies and their cast, and suppose that the total number of actresses is equal to the total number of actors. We can view it as a bipartite graph between actresses and actors, in which there is an edge iff the two have co-appeared in a movie. Prove that if this graph has a perfect matching, then there exists a winning strategy for Bob. (I.e., no matter how Alice plays, Bob can win.)

Let M be a perfect matching in the graph. Then Bob's strategy can simply be to output the matched vertex of the vertex that Alice plays. Clearly, Bob will always be able to make a valid move, and thus Bob cannot lose.

Bonus [5]: Prove the other direction, i.e., if there is no perfect matching, then Alice has a winning strategy.

Suppose Alice constructs a maximum matching M. Since it's not a perfect matching, and the total number of actors and actresses is equal, there must be an actress who isn't matched. This would be Alice's first name A_1. Bob then has to respond with a B_1 connected to A_1. If B_1 is not in the matching, then we have found an augmenting path, and thus M isn't maximum.
Otherwise, B_1 is in the matching, and thus Alice can respond by setting A_2 to be the vertex matched to B_1. Then Bob must respond with B_2 connected to A_2 (or lose!). We can make

the same argument as above, and continue this way. Thus since the game is finite, and since there is no augmenting path, Alice always wins.

4. (a) (5 points) Finding "communities" in social networks is a problem of significant interest in network science. A common problem here is the following: we are given an undirected graph $G = (V, E)$, and a target *connectivity* λ. The goal is to find a subset S of users, such that the number of edges in G both of whose end points lie in S is at least $\lambda|S|$. I.e., on average, a user in S is connected to 2λ other users in S. By reasoning about the graph in Figure 2, give a polynomial time algorithm to find such a set S, or conclude that none exists.

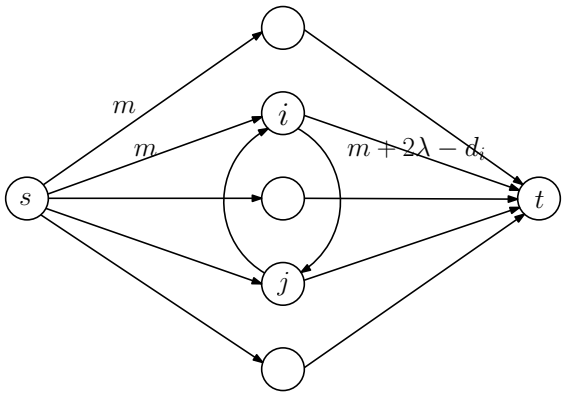

Figure 2: This graph has one node corresponding to every vertex in G, along with additional vertices s and t. The capacities of the edges to/from s, t are as shown. m is the total number of edges in G, and d_i refers to the degree of vertex i. For every edge $i, j \in E$, there are two directed edges of capacity 1.

For a given λ, let us compute the min cut between s and t. Suppose the cut splits the vertices in V and $(S, \overline{S})$. Then the total number of edges crossing the cut (in the forward direction) is precisely:

$$m(n - |S|) + \sum_{i \in S}(m + 2\lambda - d_i) + E(S, \overline{S}).$$

The three terms correspond to (a) edges out of s, (b) edges to t, and (c) edges between S and $\overline{S}$, respectively. Noting that $\left(\sum_{i \in S} d_i\right) - E(S, \overline{S}) = 2E(S, S)$ (in the undirected graph G), the above quantity simplifies to $mn + 2(\lambda|S| - E(S, S))$. Thus, if the min cut value is $\leq mn$, then it means that the cut that achieves this value satisfies $E(S, S) \geq \lambda|S|$, which is precisely the kind of set we wish to find.

(b) (10 points) Suppose we have a collection of n rectangular tiles lying on the floor, with the ith one having dimensions $a_i \times b_i$. We wish to reduce the amount of "floor space" used by these tiles by stacking them up as much as possible. However, the tiles are quite delicate, and tile j can be placed on top of tile i only if it is "fully contained" in tile i (by potentially rotating by 90 degrees), i.e., if either ($a_j \leq a_i$ and $b_j \leq b_i$) or ($a_j \leq b_i$ and $b_j \leq a_i$).

Design an algorithm that, given $\{a_i, b_i\}_{i=1}^n$, produces the stacking that leads to the minimum amount of total floor space used.

E.g.: Suppose we have tiles A, B, C, D, with dimensions $(10, 20), (15, 10), (5, 15), (10, 10)$ respectively. Then, one way to stack might be to have two stacks: first with C on top of B on top of A, and second with just D. This stacking yields a total floor space of $200 + 100 = 300$. On the other hand, having the first stack have D on top of B on top of A, and the second just C uses a total floor space of $200 + 75 = 275$.

[*Hint:* You may use the fact that the weighted version of maximum bipartite matching is solvable in polynomial time. (I.e., the version in which edges have weights and the goal is to maximize the sum of the weights of the edges in the matching.)]

Construct a bipartite graph, with each side consisting of the set of tiles. Now, place an edge between tile i and tile j, if (a) $i \neq j$, and (b) i can be placed on top of j (possibly by rotating 90 degrees). The weight of the edge is precisely the area of tile i.
Now, any matching in the bipartite graph corresponds to a placement of the tiles on the floor. (If a tile is unmatched, it is placed on the floor, and otherwise, the matching gives the tile it is to be placed on.)
Finally, the total floor space used is the total area of the unmatched tiles, which is precisely equal to the total area of the tiles, minus the weight of the matching! Thus, finding the maximum weight matching yields the desired placement of the tiles. Since we assume that the max wt matching can be found in poly time, the algorithm also runs in poly time.