CME 305: Discrete Mathematics and Algorithms

Instructor: Reza Zadeh (rezab@stanford.edu)

Problem Session #2 – 02/15/17

1. Prove that the size of a maximal matching is at least $\frac{1}{2}$ of the size of maximum matching.

   Solution: Let M^* be the maximum matching of a graph $G = (V, E)$. Suppose $|M^*| = k$ and let M be any maximal matching of G. Now, for a contradiction, suppose $|M| < k/2$. The number of vertices covered by M^* is $2k$ and M covers $< k$ vertices. That is, M does not cover at least one end point of all k edges in M^*. So there exists an edge $e = (u, v) \in M^*$ such that neither u nor v are covered by edges in M. But that means $M \cup \{e\}$ is still a matching, contradicting the fact that its maximal. So we have shown that for all maximal matchings M, $|M| \geq |M^*|/2$.

2. Given a network $G(V, E, s, t)$, describe a algorithm to determine whether G has a unique minimum $s - t$ cut.

   Solution: First compute a minimum s-t cut C, and define its volume by $|C|$. Let $e_1, e_2, \ldots, e_k$ be the edges in C. For each e_i, try increasing the capacity of e_i by 1 and compute a minimum cut in the new graph. Let the new minimum cut be C_i, and denote its volume (in the new graph) as $|C_i|$. If $|C| = |C_i|$ for some i, then clearly C_i is also a minimum cut in the original graph and $C \neq C_i$, so the minimum cut is not unique. Conversely, if there is a different minimum cut C' in the original graph, there will be some $e_i \in C$ that is not in C', so increasing the capacity of that edge will not change the volume of C', thus $|C| = |C_i|$. In conclusion, the graph has a unique minimum cut if and only if $|C| < |C_i|$ for all i. The algorithm takes at most $m + 1$ computing of minimum cuts, and therefore runs in polynomial time.

3. An $n \times n$ grid is an undirected graph consisting of n rows and n columns of vertices, as shown below. We denote the vertex in the ith row and the jth column by (i, j). All vertices in a grid have exactly four neighbors, except for the boundary vertices, which are the points (i, j) for which $i = 1$, $i = n$, $j = 1$, or $j = n$.

   Given $m \leq n^2$ starting points $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ in the grid, the escape problem is to determine whether or not there are m edge-disjoint paths from the starting points to any m different points on the boundary. For example, the grid in Figure 1 has an escape.

   Use maximum flow to give an efficient algorithm to solve the escape problem.

   Solution: To use max-flow algorithm, we need to add a source node s and a destination node t to the grid. We connect s to each of the m starting points and draw a link from every boundary point to t. We replace grid edges with two directed edges and give a capacity of one to all edges in the graph. Because all capacities are integral, the max flow will be integral, so we may find feasible edge-disjoint paths. If the size of maximum-flow in the resulting graph is m then there exists m edge-disjoint paths from each of the m starting points to a boundary point.

   Note that to ensure correctness, we need to consider the case when both directed edges between a pair of grid nodes have flow in the output max flow. In this case, the output flow is not edge disjoint. However, we may remove the flow on both these edges without

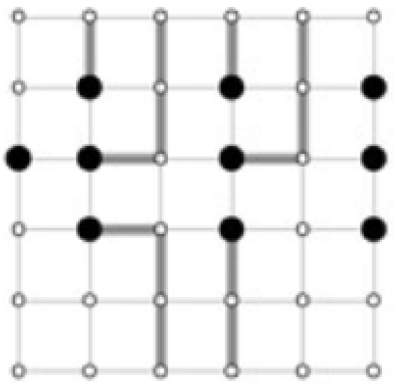

Figure 1: Grid for the escape problem. Starting points are black, and other grid vertices are white.

changing any feasibility and without affecting the size of the flow. Therefore, these cases do not affect the size of the maximum flow. To actually find the paths, we first need to remove all these bi-directional flows between grid nodes.

To find the m edge-disjoint paths, we start from s and follow the edges with positive flow to reach t, call this path P. P is clearly an escape path from one of the starting points to a boundary point. We remove edges of P from the flow network, it is easy to observe that the remaining flow is still feasible. We can do this $m-1$ more times to retrieve all the m escape paths.

4. You've periodically helped the medical consulting firm Doctors Without Weekends on various hospital scheduling issues, and they've just come to you with a new problem. For each of the next n days, the hospital has determined the number of doctors they want on hand; thus, on day i, they have a requirement that *exactly* p_i doctors be present.

   There are k doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus doctor j provides a set L_j of days on which he or she is willing to work.

   The system produced by the consulting firm should take these lists and try to return to each doctor j a list L'_j with the following properties.

   (A) L'_j is a subset of L_j, so that doctor j only works on days he or she finds acceptable.

   (B) If we consider the whole set of lists $L'_1, \ldots, L'_k$ it causes exacty p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

   (a) Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers $p_1, p_2, \ldots, p_n$, and the lists $L_1, \dots, L_k$, and does one of the following two things.

- Return lists $L'_1, L'_2, \dots, L'_k$ satisfying properties (A) and (B); or
- Report (correctly) that there is no set of lists $L'_1, L'_2, \dots, L'_k$ that satisfies both properties (A) and (B).

(b) The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no accecptable set of lists $L'_1, L'_2, \dots, L'_k$ exists.

Thus the hopspital relaxes the requirements. They add a new parameter $c > 0$ and the system now should try to return to each doctor j a list L'_j with the following properties.

(A*) L'_j contains at most c days that do not appear on the list L_j.

(B) (*Same as before*) If we consider the whole set of lists $L'_1, \dots, L'_k$ it causes exacty p_i doctors to be present on day i, for $i = 1, 2, \dots, n$.

Describe a polynomial-time algorithm that implements this revised system. It should take numbers $p_1, p_2, \dots, p_n$, the lists $L_1, L_2, \dots, L_k$, and the parameter $c > 0$, and do one of the following things.

- Return lists $L'_1, L'_2, \dots, L'_k$ satisfying properties (A*) and (B); or
- Report (correctly) that there is no set of lists $L'_1, L'_2, \dots, L'_k$ that satisfies both properties (A*) and (B).

Solution:

1. Since we have no limit on how many days a doctor is willing to work, there is a simple solution to this problem: For each day i, let q_i be the number of doctors that are willing to work on this day. If $q_i < p_i$, clearly there is no possible assignment; otherwise, just arbitrarily pick p_i from those q_i doctors to work on day i.

Lesson here: not everything is network flow. Always observe the special structure and special properties of the problem before solving it.

2. We do the same thing as in part (1). The difference is for day i suppose $q_i < p_i$, then we have a deficiency. Let $d_i = \max(0, p_i - q_i)$. Construct a network where we have one node u_i for each doctor i, and node v_j for each day j. For each doctor i add an edge (s, u_i) with capacity c, and for each day j add an edge (v_j, t) with capacity d_j. For each pair (u_i, v_j), add an edge of capacity 1 if doctor i do *not* want to work on day j. There is a feasible assignment if and only if the maximum flow in this network saturates all incoming edges of t.

The proof is simple. For each day j we need d_j extra doctors, and these doctors can only be the doctors that do not want to work on day j, and each doctor can only work on no more than c such days. Therefore if we split each doctor into c identical copies and each day j into d_j identical copies, then the problem turns into finding a bipartite matching of the doctors and the days such that all days need to be matched. The corresponding network flow formulation is what we had in the previous paragraph.

5. **Kleinberg and Tardos 11.9**

   Consider the following maximization version of the 3-Dimensional Matching Problem. Given disjoint sets X, Y, and Z, and given a set $T \subseteq X \times Y \times Z$ of ordered triples, a subset $M \subseteq T$ is a *3-dimensional matching* if each element of $X \cup Y \cup Z$ is contained in at most one of these triples. The *Maximum 3-Dimensional Matching Problem* is to find a 3-dimensional matching M of maximum size.

   Give a polynomial-time algorithm that finds a 3-dimensional matching of size at least 1/3 times the maximum possible size.

   Solution:

   Each matching edge is a triangle in a 3D-matching. Similar to a maximum bipartite matching, we approximate maximum 3D-matching with a maximal 3D-matching. We start by adding triangles until we can't add any more of them. Maximal 3d-matching is a 3-approximation because each edge in the maximal matching could have replaced three edges in the maximum matching (each endpoint of a triangle could be part of a different triangle in maximum 3D-matching).

6. MAXCLIQUE is the problem of finding the maximum size clique in an undirected graph $G = (V, E)$. Consider a version of this problem such that given a graph G and number k, we want to output if G has a clique of size at least k. Show that this problem is NP-complete.

   Solution: The problem as stated is a decision problem. Suppose we have a yes certificate (list of k nodes) for a given graph, then to check that it is correct, we need to check that all edges exist, which takes $O(k^2)$ time. So the problem is in NP.

   Now to show the problem is NP-hard, we will reduce the decision version of INDEPENDET SET (does a graph contain an independent set of size at least k) to our problem. Suppose we have a polytime algorithm for MAXCLIQUE and and input graph $G = (V, E)$ and k for the independent set problem. Consider running our MAXCLIQUE algorithm on the graph $\bar{G}$ and k.

   Computing $\bar{G}$ from G takes $O(n^2)$ time. We must now show G contains an independent set of size $\geq k$ if and only if $\bar{G}$ contains a clique of size $\geq k$. But this fact follows by definition, so MAXCLIQUE is NP-hard and therefore NP-complete.

7. Recall that an instance of 3-SAT consists of formula in conjunctive normal form (CNF) where each clause has 3 literals each,

   $$(X_1 \vee \bar{X}_3 \vee X_4) \wedge (\bar{X}_1 \vee X_2 \vee X_5) \ldots$$

   In *Not-All-Equal* 4SAT (NAE4SAT), there are 4 literals per clause, and an instance is satisfiable iff there exists an assignment of values to variables such that there is at least one T and one F literal in each clause.

   Show that NAE4SAT is NP-complete by a reduction from 3SAT.

Solution: I found the problem here: `http://www.cs.cornell.edu/courses/cs4820/2014sp/notes/reduction-maxcut.pdf`, which uses this fact as part of a larger reduction showing that MAX-CUT is NP-hard.

Note that I also went through Thm 6.6 shown here `https://www.cs.ucsb.edu/~suri/cs231/ProbMethod.pdf` - I believe the difficulty of this problem is higher than an exam question, but thought the techniques in the proof might be useful.