CME 305: Discrete Mathematics and Algorithms

Instructor: Reza Zadeh (rezab@stanford.edu)

Midterm Review Session 02/10/15

Note that these solutions are compact and only provide the key ideas in answer of the question. They should not be considered model solutions. More complete answers are expected on homeworks and in exams.

1. Hamiltonian Paths and Cycles

   (a) Show that determining whether a graph contains a Hamiltonian Path is at least as hard as determining whether a graph contains a Hamiltonian Cycle

   Solution: A Hamiltonian Path is a path that exhausts all the edges. Given a black box for HP we want to show that we can solve HC. Fix some node i, if there is a HC then there must exist a HP between node i and (at least) one of its neighbors j. For a given neighbor j delete the edge (i, j). Also delete all but one edge incident to i and all but one edge incident to j, call the remaining edges a and b. For each choice of a and b check if there is a HP. Such a path must start at i and end at j. If you find such a path after iterating over all neighbors j of i then a HC exists. Else a HC does not exist.

   (b) Show that determining whether a graph contains a Hamiltonian Cycle is at least as hard as determining whether a graph contains a Hamiltonian Path

   Solution: Given a black box for HC we want to show that we can find HP. Add a new node α to the graph, add edges (α, i) for every node i in the original graph. If there is a HC on the new graph then there must exist a HP on the original graph.

   (c) Suppose you have a black box that takes input a graph and gives output YES if and only if the graph contains a Hamiltonian Cycle. Show how to use this black box to find a Hamiltonian Cycle.

   Solution: Check if the graph contains a HC, if it does not stop and return NO. Consider edge (i, j). As there is a HC, it must traverse two of the edges incident with j. Delete all but two of the edges incident with j: keep edge (i, j) and one other edge a. Call to the black box. If the black box returns YES then edges (i, j) and a are in the HC. If the black box returns no, iterate through the other edges a. If the black box returns NO for all edges a then edge (i, j) is not in the HC.

2. Effective Resistances

   For each of the following graphs determine the effective resistance between the given pairs of nodes exactly, and determine bounds on the covering time of the graph.

   (a) Lollipop graph with node i in the clique and node j in the stick.

   Solution: Let node i be in the clique, node k be at the base of the stick and node j be at the tip of the stick. Then by the series rule $R_{ij} = R_{ik} + R_{kj}$. By the series rule we can see $R_{kj} = n/2$. So we must now analyze R_{ik}.

Within the clique there is 1 path direct from i to k and $(n-4)/2$ distinct paths from i to k via some other node (there are edges between these intermediate nodes, however we ignore them - this can not decrease the effective resistance hence our result will be an upper bound). By the parallel rule:

$$R_{ik} = \frac{1}{1/1 + 1/2 + \cdots + 1/2} = \frac{1}{1 + \sum_{\ell=1}^{(n-4)/2} 1/2} = \frac{4}{n}$$

(b) Barbel graph with nodes i and j in separate cliques.

Solution: Let node i be in the first clique, node k_1 be in the first clique and adjacent to node k_2 in the second clique. Let node j be in the second clique. Then by the series rule $R_{ij} = R_{ik_1} + R_{k_1k_2} + R_{k_2j}$. Clearly $R_{k_1k_2} = 1$. The analysis of R_{ik_1} and R_{k_2j} follows in same pattern as for the clique in the lollipop graph giving:

$$R_{ij} = \frac{4}{n} + 1 + \frac{4}{n}$$

(c) Cycle graph with effective resistance as a function of the distance between i and j.

Solution: Let d ($1 \leq d \leq n/2$) denote the distance between i and j. There are two paths from i to j, one of length d and the other of length $n-d$. Hence by the parallel rule:

$$R_{ij} = \frac{1}{\frac{1}{d} + \frac{1}{n-d}} = \frac{d(n-d)}{n}$$

This can be differentiated with respect to d to find the distance that maximizes the effective resistance between any nodes i and j in the cycle.

3. Integrality Problems as Max Flow

These problems take 3 stages:

(a) Construct a flow problem

(b) Show that a certain (maximum) flow is possible on the graph

(c) Relate the solution from Ford-Fulkerson to the original problem

(a) Several families go out to dinner together. To increase their social interaction, they would like to sit at tables so that no two members of the same family are at the same table. Assume that the dinner contingent has p families and that the ith family has q_i members. Assume also that there are t tables, and that the jth table has a seating capacity of b_j. Show how to find a satisfying assignment of people to tables in polynomial time.

Solution: Construct the graph as follows: Create one node for each family i and create one node for each table j. Create a source node s and a sink node t. Let there be edges (i, j) for all choices of i and j with capacity 1. Let there be edges (s, i) for all choices of i with capacity q_i. Let there be edges (j, t) for all choices of

j with capacity b_j. We have transformed this problem into a flow problem with integer capacities which can be solved (with integer flows) by Ford-Fulkerson algorithm.

Part 2 is not required for this problem. We must assume that the problem is feasible.

Given a solution to the flow problem. If edge (i, j) is saturated in the flow problem then family i sends one person to sit at table j.

(b) You are given an $N \times N$ matrix A such that each entry of the matrix is a non-negative number. Further, the sum of the entries in any row or column is an integer. You are allowed to round each fractionally entry in the matrix i.e. to change each non-integer entry to either the next higher or next lower integer. Prove that there is a way of rounding each entry such that the row and column sums remain unchanged.

Solution: We carry out some preprocessing before converting to a flow problem. Begin by removing all the integer parts from every entry in the matrix A. So $a_{ij} \in [0, 1)$. As we have only removed integer quantities the row and column sums are still integer. Let k_i denote the sum of the ith row and k_j denote the sum of the jth column. Then we need to choose k_i entries in the ith row to be 1 and the remaining entries to be zero - in such a way that when we do this for all rows i the corresponding column constraints are satisfied.

Create one node for each row i and create one node for each column j. Create a source node s and a sink node t. Let there be an edge (i, j) with capacity 1 if we can put a 1 in entry (i, j) of the matrix (so $a_{ij} \neq 0$). Let there be edges (s, i) for all choices of i with capacity k_i. Let there be edges (j, t) for all choices of j with capacity k_j. We have transformed this problem into a flow problem with integer capacities which can be solved (with integer flows) by Ford-Fulkerson algorithm.

In order for the flow to satisfy the original problem we need to show that the maximum flow is $\sum_i k_i = \sum_j k_j$. Saturate every edge (s, i) and (j, t), let the flow along edge (i, j) be equal to a_{ij}. This is a feasible (non-integer) flow that has value $\sum_i k_i$. If we consider the cut $S = \{t\}$ then we have a cut that obtains the same value as this flow. Hence (by strong duality) both the flow and cut are optimal.

Given a solution to the flow problem. If edge (i, j) is saturated then we set $a_{ij} = 1$ otherwise we set $a_{ij} = 0$.

4. Greedy Max-Cut

Consider the greedy algorithm that, given an ordering $v_1, \ldots, v_n$ of the vertices, assigns v_1 to set A, then greedily partitions the other vertices (by sequentially assigning each unassigned vertex v to either A or B according to whether v has more neighbors already assigned to B or more neighbors already assigned to A.) Assume that ties are broken by assigning the point to set A. Prove that the cut found by this greedy algorithm cuts at least $(|E| + |B|)/2$ edges, where $|B|$ is the size of set B at the end of the algorithm.

Solution: The solution to this algorithm mixes ideas from randomized and deterministic algorithms. This approach is not a major focus on the course. We give the key intuition below.

Consider pausing the algorithm when it is considering assigning the ith node. We want to compare three possible assignments:

$$\begin{aligned}
&\text{cuts}(v_1,\ldots,v_{i-1}) + \text{cut}_{\text{random}}(v_i) + E\text{cuts}_{\text{random}}(v_{i+1},\ldots,v_n)\\
&\text{cuts}(v_1,\ldots,v_{i-1}) + \text{cut}_{v_i\in A}(v_i) \quad + E\text{cuts}_{\text{random}}(v_{i+1},\ldots,v_n)\\
&\text{cuts}(v_1,\ldots,v_{i-1}) + \text{cut}_{v_i\in B}(v_i) \quad + E\text{cuts}_{\text{random}}(v_{i+1},\ldots,v_n)
\end{aligned}$$

Each line gives the number of edges in the cut given that nodes $v_1,\ldots,v_{i-1}$ have already been assigned, assuming nodes $v_{i+1},\ldots,v_n$ will be randomly assigned. The thing that differs between the three lines is how node v_i is assigned.

By expectations the average of the 2nd and 3rd lines must be equal to the random assignment. So if we deterministically assign a node to B then the third line must be greater that the first line by at least $1/2$. Over every assignment to B we accumulate $|B|/2$ worth of these small improvements over the randomized algorithm. Hence the deterministic algorithm must achieve at least $|B|/2$ better than the random algorithm. Hence $|E|/2 + |B|/2 = (|E| + |B|)/2$ is a lower bound on the cut size.

5. K-SAT

Suppose you have a satisfiability problem where every clause contains at most K literals ($K > 3$). Reduce this problem to 3-SAT. Explain why in general this problem can not be reduced to 2-SAT.

Solution: Consider the clause $(A \cup B \cup C \cup D)$ we can break it into two clauses of size 3 as follows:

$$(A \cup B \cup C \cup D) \equiv (A \cup B \cup \alpha) \cap (\bar{\alpha} \cup C \cup D)$$

where α is a new literal. If $C \cup D$ is true then α is true so both clauses are satisfied. If $A \cup B$ is true then α is false so both clauses are satisfied. If $A \cup B \cup C \cup D$ is false then no choice of α can satisfy both clauses. Hence the new pair of clauses is equivalent to the original clause.

The above idea can be applied iteratively:

$$(A \cup B \cup C \cup D \cup E \cup F) \equiv (A \cup B \cup \alpha) \cap (\bar{\alpha} \cup C \cup \beta) \cap (\bar{\beta} \cup D \cup \gamma) \cap (\bar{\gamma} \cup E \cup F)$$

So a clause with K literals can be replaced by $K - 2$ clauses each with 3 literals.

Why can we not reduce to 2-SAT? Because we have to add a new literal every time we break apart an old clause. So some clauses must contain two of the new literals (α, β, γ) and one of the original literals.