CME 305: Discrete Mathematics and Algorithms

Instructor: Professor Amin Saberi (saberi@stanford.edu)

Midterm – 02/17/11

This exam is closed notes/books/laptops. You will have until the end of class to ask any questions clarifying any of the problems. You may then set aside 3 more hours to work on the exam individually. The exam is due at 8pm today. Submit your work into the CME305 dropbox in the basement of Huang located across from room 041 (next to the HP Garage).

Problem 1. An *edge cover* C of a graph $G(V, E)$ is a subset of E such that for all $v \in V$ there exists $e \in C$ with $v \in e$ i.e. an edge set covering all vertices. Let C^* be a minimum edge cover, that is $|C^*| \leq |C|$ for all edge covers C of G. Prove that

$$|C^*| + |M^*| \leq |V|$$

where M^* is a maximal matching of G.

Solution: Let S be the set of vertices not covered by M^*. Note that S is an independent set. Let C be all edges in M^* plus edges which connect M^* to S. This is an edge cover. Then,

$$|C^*| \leq |C| = |M^*| + |S| = |M^*| + (|V| - 2|M^*|) = |V| - |M^*|$$

and rearrange.

Problem 2: Given a graph $G(V, E)$ we want to partition the vertices of the graph into two (disjoint) sets A and B (i.e. $A \cup B = V$ and $A \cap B = \varnothing$) such that the number of edges with one endpoint in A and the other in B is maximized. In other words, we are looking for the maximum cut (A, B) of the graph.

Consider the following algorithm. We start with an arbitrary partition (A, B). If there exists a vertex $v \in V$ such that moving it to the other set in the partition increases the number of edges in the cut, do so. Proceed until there is no such vertex.

(a) Show that the algorithm stops.

(b) Show that when the algorithm stops, the size of the cut is at least half of the optimum.
Hint: Consider the neighbors of each vertex after the algorithm stops.

Solution:

1. After each iteration, the number of edges in the cut increases by at least one. Obviously $|E|$ is upper bound on the number of cutting edges. Thus the algorithm will stop after at most $|E|$ steps.

2. Suppose the algorithm terminates and gives us a cut (A, B). For each vertex $v \in V$, denote $d(v)$ as the degree of v. Moreover, denote $d_I(v)$ as the number of edges of the form (v, u) such that u is in the same partition of v. Similarly denote $d_E(v)$ as the number of the edges of the form (v, u) such that u is in the different partition from v. Obviously $d(v) = d_I(v) + d_E(v)$.

   When the algorithm terminates, we have $d_E(v) \geq d_I(v)$ for all $v \in V$. Otherwise we can move some v to the other partition to give a larger cut. Denote OPT as the optimal cut and C as the cut given by the algorithm. Then

$$OPT \leq |E| = \frac{1}{2}\sum_{v\in V} d(v) = \frac{1}{2}\sum_{v\in V} d_I(v) + d_E(v) \leq \sum_{v\in V} d_E(v) \leq 2C. \tag{1}$$

Problem 3: Consider the following problem: Given n items with sizes $a_1, a_2, \cdots a_n$ all in $(0, 1]$, find a packing in unit size bins that minimizes the number of bins used.

(a) Prove that the following algorithm is a factor 2 approximation: Consider the items in an arbitrary order. In the i^{th} step, suppose you have a list of partially packed bins, say $B_1, B_2, \ldots, B_k$. If possible, put a_i into any one of them. If a_i does not fit into any of these bins, open a new bin B_{k+1} and put a_i in it.

(b) Give an example on which the above algorithm does at least as bad as 5/3 of OPT, where OPT is the number of bins in the optimal packing.

(c) Consider a modification of the algorithm in part (a). At the i^{th} step, suppose you have a list of partially packed bins, say $B_1, B_2, \ldots, B_k$. You may only put a_i into bin B_k. If a_i does not fit into bin B_k, open a new bin B_{k+1} and put a_i in it. Prove that this modified algorithm also gives a factor 2 approximation.

Solution:

(a) Observe that no two bins can be less than half packed. Otherwise, the algorithm would have put those items into one bin.

   Denote b_i as the used amount in box B_i. Then $b_i \geq 1/2$ except for one b_j.

$$OPT \geq \sum_{i=1}^{n} a_i = \sum_{i=1}^{k} b_i > \frac{k-1}{2}$$

   Thus $2OPT > k - 1$, or $OPT \geq k$ because both OPT and k are integers.

(b) The first 6 items have size 0.1 (algorithm packs them into 1 bin with 0.4 left). The next 6 items have size 0.35 (algorithm packs them into 3 bins with 0.3 left in each). The next 6 items have size 0.55 (algorithm has to pack them into 6 new bins). So the total number of bins used by the algorithms is $1 + 3 + 6 = 10$. But the optimal packing is $0.55 + 0.35 + 0.1 = 1$ in each of the OPT= 6 bins. The ratio is $10/6 = 5/3$.

(c) Let OPT be the optimal number of bins and r be the number of bins used by our algorithm. Let w_j be the weight of the packing of bin j produced by our algorithm. By the definition of the algorithm we have $w_{i-1} + w_i > 1$ for all $i = 2, \ldots, k$. If k is even then

$$OPT \geq \sum_{i=1}^{n} a_i = \sum_{i=1}^{k} w_i = \sum_{i=1}^{k/2} w_{2i-1} + w_{2i} > \frac{k}{2}$$

Thus $k < 2OPT$. If k is odd, then

$$OPT \geq \sum_{i=1}^{n} a_i = \sum_{i=1}^{k} w_i > \sum_{i=1}^{(k-1)/2} w_{2i-1} + w_{2i} > \frac{k-1}{2}$$

Thus $k - 1 < 2OPT$, then $k \leq 2OPT$ because both of them are integers.

Problem 4: A *vertex coloring* of a graph $G(V, E)$ is an assignment colors to each vertex of a graph such that no edge connects two identically colored vertices. A graph G is called k-vertex-colorable if and only if there exists a vetex coloring of G with k (or less) colors.

Give a polynomial time algorithm for coloring the vertices of a 3-vertex-colorable graph of size n with at most $O(\sqrt{n})$ colors.

Hint: Try coloring the neigborhood of some vertex.

Solution: First, we first prove two claims.

1. For each vertex v, its neighborhood $N(v)$ is a bipartite graph (thus 2-colorable).

   Proof: Assume $N(v)$ is not bipartite. Then it has an odd cycle with each vertex in the cycle connected to v. It is easy to see that such a graph is not 3-colorable which contradicts the main assumption of this problem.

2. Any graph with maximum degree Δ is $\Delta + 1$-colorable.

   Proof: The coloring is produced with a greedy strategy. For each uncolored vertex we assign distinct colors to $v \cup N(v)$ which has a maximum cardinality $\Delta + 1$. This strategy will never run into a conflict since at each step we have at most $\Delta + 1$ vertices to color and that many total colors to choose from.

We now state the algorithm to produce a $O(\sqrt{n})$-coloring for any 3-colorable graph G.

1) Partition the vertices V of the graph in the set $S = \{v \in V | d(v) \geq \sqrt{n}\}$. For each $v \in S$ use colors c_{v1} for v and c_{v2}, c_{v3} for its neighborhood $N(v)$ (by claim (a)).

2) Color the remaining vertices with exactly $\sqrt{n}$ colors. This can be done by a greedy procedure from claim (b) since the maximum degree of the vertices in $V - S$ is $\sqrt{n} - 1$.

Clearly this algorithm runs in polynomial time. There are at most $\sqrt{n}$ vertices in S (since there are n total). And we use three colors at each step when coloring S and their neighbors. The remaining vertices are shown to be $\sqrt{n}$-colorable. Thus the graph can be colored with $O(\sqrt{n})$ colors.