CSE 208A Homework 1
Randomized algorithms
Due Thursday, May 24

For each of the algorithm problems, design as asymptotically efficient an algorithm as possible. Give a correctness argument (explanation, if it is relatively simple, or proof if not) and time analysis. You may use any well-known algorithm or data structure, or algorithm from the text or from class, as a sub-routine without needing to provide details.

Vazirani, Ex. 1.1, page 7

Following the hint, arbitrarily label the nodes $1, ...n$, and construct two sets of edges: $F = \{(u, v)|$ u's label is smaller than $v's\}$,B= $\{$(u,v)—$u'slabelislargerthan$v's$\}$ B and F can be constructed in $O(m)$ time, and the algorithm returns the one of higher total weight.

We claim F and B are both acyclic. If there is a cycle in F, we can without loss of generality let u_1 be the node in the cycle of largest label, and let u_2 be the next node in the cycle. But then there must be an edge from u_1 to u_2, so u_2 must have a larger label than u_1, contradicting the choice of u_1. Similarly for B.

Let E' be the set of non-self-loop edges. $E' = B \cup F$. Note that self-loops cannot be in any acyclic sub-graph. So any acyclic sub-graph is a subset of E', and so the largest acyclic subgraph has size $\leq |E'|$. Since $E' = B \cup F$, the larger of B, F has size $\geq 1/2|E'|$. Therefore, the approximation ratio of the algorithm is at least $1/2$.

Vazirani, Ex. 2.2, page 23 Let $m = |E|$. No cut can have more than m edges. Let C_i be the cut of the local search algorithm after i steps. Because each step of the local search algorithm, increases the size of the cut by at least 1, $m \geq C_i \geq i$. Therefore, the local search algorithm always terminates in at most m steps.

When the local search algorithm terminates, the resulting solution is a local optimum, in that no single swap increases the number of cut edges. Let C be the size of the cut at a local optimum. For each $v \in V$, let s_v be the number of edges between v and nodes on the same side of the cut, and let c_v be the number of edges adjacent to v in the cut. Swapping v to the other side of the cut changes c_v to s_v, and vice versa; since swapping v does not increase the cut, $s_v \leq c_v$ for every v.

Then $\sum_{v \in V} s_v \leq \sum_{v \in V} c_v$. Each edge not in the cut contributes two to the left-hand side and each edge in the cut contributes 2 to the right-hand side, for their two endpoints. Therefore, $2(m - C) \leq 2C$ and $C \geq m/2 \geq 1/2 MaxCut$. Therefore this algorithm achieves approximation ratio $1/2$. (Like many others we've seen!)

Naive implementation would do an exhaustive search over nodes v, computing s_v and c_v for each. This would take time $O(m)$ per stage, and hence total time $O(m^2)$. Slightly better is to maintain s_v and c_v, as well as a single bit for the side of the cut v is on, for each node v. Since swapping u only changes s_v or c_v by one if v is a neighbor of u, after a swap we can run through neighbors of the swapped node, and update the values in constant time per neighbor. Then we can see if any node has $s_v > c_v$ in $O(n)$ time so each stage takes $O(n)$ time and the whole algorithm is $O(nm)$.

Vazirani, Ex. 2.15, page 25 Let $w(S) = \sum_{u \in S} w(u)$. Let $B_1, ..B_k$ be the k sets chosen by the greedy heuristic, and let $A_1, ..A_k$ be the optimal cover. Let $A = \cup_{i=1}^{i=k} A_i$. We prove by induction on j that $w(\cup_{i=1}^{i=j} B_i) \geq (1-(1-1/k)^i)w(A)$; the desired approximation ratio follows from the case $j = k$.

For $j = 1$, we prove that $w(B_1) \geq 1/kw(A)$. Since B_1 is the set of largest weight, $w(A) \leq \sum_{i=1}^{i=k} w(A_i) \leq kw(B_1)$.

Assume the claim for j. Let $B' = \cup_{i=1}^{i=j} B_i$. Then by our choice of B_{j+1}, $w(B_{j+1} - B')$ is the largest of any set. So $w(A - B') \leq \sum_{i=1}^{i=k} w(A_i - B') \leq kw(B_{j+1} - B')$. Thus $w(\cup_{i=1}^{i=j+1} B_i) = w(B') + w(B_{j+1} - B') \geq w(B') + 1/k(w(A - B')) \geq (1 - 1/k)w(B') + 1/kw(A) \geq (1 - (1 - 1/k)^j)(1 - 1/k)w(A) + 1/kw(A) = ((1 - 1/k) - (1 - 1/k)^{j+1} + 1/k)w(A) = (1 - (1 - 1/k)^{j+1})w(A)$ as claimed.

Although the problem doesn't ask for it, let's also give an implementation and time analysis. The naive algorithm would do an exhaustive search over the m possible sets, and for each count the uncovered elements, a total of $O(nmk) = O(m^2 n)$ time. Say we have m sets on n elements each. We can create a bipartite graph between sets and elements, with $E = \sum_i |S_i| \leq mn$ edges representing membership. We can maintain for each set the total weight of uncovered elements , and keep these in a max heap with double links to the corresponding set. For each element, keep a boolean saying whether it is covered. Choose and delete the max element of the heap. For each uncovered element x and each set that contains x, subtract $w(x)$ from the set's weight and readjust the heap. Note that over the course of the algorithm, each element is covered at most once, so there are at most E heap adjustments. So the total time for all heap operations, and hence the algorithm, is $O((E+k)\log m) = O(mn \log m)$, since $k \leq m$. (If the sets are small, ie. $|S_i| \leq c$ for some constant c, it could be much faster, but the naive implementation will be somewhat faster too.)

The independent set problem, is: given an undirected graph, find the largest independent set S of vertices, i.e., a set so that no adjacent vertices x, y are both in S. A. Give a linear programming relaxation for the independent set problem. (2 points) B. Give a poly-time algorithm that, given a graph G with n vertices, either finds an independent set of size at least $n/3$ or certifies that no independent set in G has size greater than $2n/3$. (It may involve using an algorithm for linear programming.) (6 points) C. Consider the special case of graphs of at most degree 3. Find $\alpha > 1/3$ and a poly-time algorithm that either finds an independent set of size αn or certifies that no independent set has size $(1-\alpha)n$. (2 points + 2 extra credit if you beat my α.)

A. A linear programming relaxation introduces x_i for each node i, constraints $0 \leq x_i \leq 1$ and $x_i + x_j \leq 1$ for each edge $\{i,j\}$ and objective $\sum_{i=1}^{i=n} x_i$. Note that we can always achieve $n/2$ by setting each $x_i = 1/2$. We'll call this the trivial solution to the relaxed problem.

B. The idea is, that from a very non-trivial solution to the linear-programming relaxation, we can recover a big independent set. The algorithm is: Use a linear programming algorithm such as ellipsoid method to find an optimal solution $x_i = v_i$ to the linear programming relaxation above.

If $V = \sum_i v_i < 2/3n$, return "No independent set is of size 2/3 n". This must be true, since any independent set I provides a solution to the relaxed problem with objective $|I|$ by setting $x_i = 1$ if $i \in I$, 0 otherwise.

If not, then $V = \sum v_i \geq 2/3n$. Let I be the set of nodes with $v_i > 1/2$. The algorithm returns I. I is an independent set since if $\{i,j\} \in E$, $v_i + v_j \leq 1$, so at least one is less than $1/2$. On the other hand, $2/3n \leq V \leq |I| + 1/2(n - |I|) = 1/2n + 1/2|I|$ so $|I| \geq n/3$.

Note that we can also do this without linear programming at all. Take a maximal matching in G. If the matching has size $n/3$ or larger, there cannot be an independent set of size $2n/3$. Otherwise, return the set of (at least $n/3$) nodes not covered by the matching.

C. We can use the LP relaxation to get a $2/3 - 1/\log n$ approximation algorithm for independent set for graphs of degree at most 3. Consider the following algorithm: Assume G is connected. Solve the LP relaxation, to get a relaxed solution v_i with $V = \sum v_i$. Delete the node of smallest v_i. From now on, there will always be a node of degree at most 2. Repeat until the graph is empty: Pick the node i of degree at most 2 that maximizes v_i. Put i in the independent set. Delete i and its at most two neighbors.

We claim that this algorithm returns an independent set of size at least $(2/3)(1-1/n)V$. Let V' be $V - v_i$, where i is the node initially dropped. Then $V' \geq V(1-1/n)$ since v_i was the smallest weight. For the analysis, divide the algorithm into two stages: During the first stage, the node put in the independent set always has $v_i \geq 1/2$; the second stage begins the first time when $v_i < 1/2$. Let G_R be the subgraph of undeleted nodes at the beginning of the second stage, let n_R be the number of nodes in

G_R and let $V_R = \sum_{j \in G_R} v_j$. Let I_1 be the part of the independent set constructed in the first stage, and I_2 that in the second stage.

First, we claim $|I_1| \geq 2/3(V' - V_R)$. Every time we put a node i in I_1, consider its at most two neighbors j, k. $v_i + v_j \leq 1$, and $v_k \leq 1 - v_i \leq 1/2$, so $v_i + v_j + v_k \leq 3/2$. So every time we put one node in I_1, we reduce the remainin V by at most 1.5. The claim follows.

Second, $|I_2| \geq n_R/3$. Every time we put a node in I_2, we delete at most 3 nodes from the graph.

The tricky part is this claim: $V_R \leq n_R/2$, the residual relaxed solution at the start of the second stage is no better than the trivial solution. Let B_R be the set of nodes in G_R with $v_i > 1/2$. No element of B_R has degree less than 3, so all such nodes have degree 3. Let S_R be the neighbors of B_R; note that each element $j \in S_R$ has $v_j < 1/2$, so in particular S_R is disjoint from B_R. Finally, let H_R be the remaining nodes in G_R; each element of $j \in H_R$ has $v_j \leq 1/2$. For $i \in B_R$, let $\delta_i > 0$ be such that $v_i = 1/2 + \delta_i$ and for each $j \in S_R$, let $v_j = 1/2 - \gamma_j$. Then $V_R \leq 1/2 n_R + \sum_{i \in B_R} \delta_i - \sum_{j \in S_R} \gamma_j$. Note that for each $j \in S_R$ connected to $i \in B_R$, $\gamma_j \geq \delta_i$. Now since each $i \in B_R$ is connected to exactly 3 $j \in S_R$, $\sum_{i \in B_R} \delta_i = 1/3 \sum_{i \in B_R} \sum_{j \text{ connected to } i} \delta_i \leq 1/3 \sum_{i \in B_R} \sum_{j \text{ connected to } i} \gamma_j = 1/3 \sum_{j \in S_R} \sum_{i \text{ connected to } j} \gamma_j \leq 1/3 \sum_{j \in S_R} 3\gamma_j = \sum_{j \in S_R} \gamma_j$. Thus, $V_R \leq n_R/2 + \sum_i \delta_i - \sum_j \gamma_j \leq n_R/2$.

Putting these pieces together, $|I_1| + |I_2| \geq 2/3(V' - V_R) + n_R/3 \geq 2/3(V' - V_R) + 2/3 V_R = 2/3 V' \geq 2/3(1 - 1/n)V$. So the above algorithm finds an independent set at least $2/3 - 1/n$ of the optimal relaxed solution If G is not connected, then we can use the above method for each connected component of size $\log n$ or greater, and solve the smaller components by brute force. This gives a general $2/3 - 1/\log n$ approximation algorithm for independent set for graphs of degree 3.

In particular, either we can conclude that there is no independent set of size greater than $(.6 + 1/\log n)n$ or we find one of size $(.4 - 1/\log n)n$.