Rishad Rahman (Deepak's Recitation) May 1, 2015

6.046 Problem 9-1

Collaborators: *None*

(a) Let $v_1 = s_1 = 1$, $v_2 = k+1$, $s_2 = k+2$, and $B = k+2$. Then the algorithm returns a value of $v = 1$ while $v_{Opt} = k+1 > kv$ so this algorithm does not guarantee a constant approximation.

(b) Let us look at

$$v_i' = \frac{B - \sum_{j=1}^{i-1} s_i}{s_i} v_i \leq v_i$$

which is in essence how much of item i, although fractional, we can put in our set. We see that

$$\sum_{j=1}^{i-1} v_j + v_i' \geq v_{Opt}' \geq v_{Opt}$$

where v_{Opt}' refers to the fractional knapsack problem i.e. we remove the integer constraints. This follows since in the fractional knapsack problem, since we already ordered everything by non-increasing order of density, we take as much of the leading terms as possible since they give the most profit for their size. Hence

$$\sum_{j=1}^{i-1} v_j + v_i \geq v_{Opt}$$

so the maximum of either term we are interested in must be $\geq \frac{v_{Opt}}{2}$.

(c) $S_{i,v} = \min\{S_{i-1,v-v_i} + s_i, S_{i-1,v}\}$. The base cases are $S_{i,j} = \infty$ if $i = 0, j \neq 0$ and $S_{i,j} = 0$ if $j = 0$. Correctness follows easily because at each step we can include a_i or not. The number of subproblems is $n^2 V$ and each problem takes $O(1)$ time so the running time is $O(n^2 V)$. We can solve the Knapsack problem exactly using this algorithm by searching for the largest $v \ \exists \ i$ such that $S_{i,v} \leq B$. The search time is also $O(n^2 V)$ hence the total running time is $O(n^2 V)$.

(d) Let Opt' be the optimal set we find in the scaled problem. It follows that

$$v_C \geq \sum_{v' \in Opt'} \frac{\epsilon V}{n} v' \geq \sum_{v \in Opt} \frac{\epsilon V}{n} \left\lfloor \left(\frac{v}{V}\right)\left(\frac{n}{\epsilon}\right) \right\rfloor \geq v_{Opt} - |Opt| \frac{\epsilon V}{n} \geq v_{Opt} - \epsilon V$$

$$\Rightarrow v_{Opt} \leq v_C + \epsilon V \leq v_C(1+\epsilon)$$

where $V \leq v_C$ since otherwise

$$\sum_{v' \in Opt'} v' \leq \sum_{v \in C} \left(\frac{v}{V}\right)\left(\frac{n}{\epsilon}\right) = v_C \frac{n}{\epsilon V} < \frac{n}{\epsilon} = V'$$

when $\frac{n}{\epsilon}$ is an integer which is clearly possible. Therefore we have a $1+\epsilon$ approximation and the runtime is dominated by Alg_3 on parameters $V' = \frac{n}{\epsilon}$ which gives $O(\frac{n^3}{\epsilon})$.

Rishad Rahman (Deepak's Recitation) May 1, 2015

6.046 Problem 9-2

Collaborators: *Jon Lu*

(a) Suppose for the sake of contradiction we had a cycle C after the reversal and that wlog T has at least one cycle initially (no cycles is trivial). C must have been a result of the reversal otherwise A did not touch C. Let C before the reversal be C'. Before proceeding with the proof, we first note that the hint follows easily since if e always appears multiple times, then then we could remove it from A which would not make it minimal.

Proof: Suppose $C = c_1 \to c_2 ... \to c_n \to c_1$. WLOG suppose $c_1 \to c_2$ was the result of a reverse so $C' = c_1 \leftarrow c_2$ ... Since $c_2 \to c_1 \in A$, we must have that there is some path P_{12} such that $c_1 \xrightarrow{P_{12}} c_2$ that does not contain any edge in A. We can do this for all instances of $c_{i-1} \leftarrow c_i$ i.e. there exists a path from c_{i-1} to c_i that does not share any edges with A. Note that instances of $c_{i-1} \to c_i$ are $\notin A$ and instances of $c_i \to c_{i-1}$ can be replaced by a path from $c_{i-1} \to c_i$ which does not contain any edges in A. Doing this for all edges in C' generates a cycle that does not contain any edges in A, a contradiction.

(b) We instead look for the minimum cycle cover which immediately solves our problem. We iterate through the edges of T and find an edge which belongs to $> k$ triangles, flip it, and decrease k by 1. We do this since that edge must be in our cover otherwise we would require $> k$ edges to cover the triangles and the flipping removes the cycles the edge belongs in from (a). We do this until we have $\leq k$ triangles for all edges. Now we claim that now we can remove any vertex v not belonging to a triangle. Let C be a cycle of T containing v_1. We claim that v_1 must belong to a triangle, we will prove this by induction on the number of vertices. The statement is trivial for 3 vertices. Suppose it is true for k vertices. Let the $k+1$ cycle be $(v_1, v_2, ..., v_k, v_{k+1})$. If the edge (v_1, v_k) exists we have the triangle (v_1, v_k, v_{k+1}) so suppose otherwise it was (v_k, v_1) instead. But then we have the k-cycle $(v_1, v_2, ..., v_k)$ which contains a triangle that includes v_1 by hypothesis. This implies none of the edges around v must be in the cycle cover otherwise v would be in a cycle. We now have our kernel which now has every vertex contained in at least one triangle and every edge is in at most k triangles. This means the maximum number of vertices occur when we have k disjoint groups of $k+2$ vertices, any more groups would mean a cycle cover of length greater than k, hence the number of vertices is most $k(k+2)$.

(c) The reduction above takes time $O(n^3k) = O(n^5)$ because we iterate through all edges and then check the number of triangles and we may have to do this up to k times. We first brute force through all 2^{k^2+2k} possible subgraphs of the kernel and check which ones are cycles in $O(k^2)$ per subgraph. Then brute force through $\binom{k^2+2k}{k}$ possible choices for a cycle cover of size k and check if any cover all the cycles which takes $O(k^2)$ time per cycle. Since the kernel is a reduction of the original problem, this solves the original problem and we get the run time is $O(n^5 + \binom{k^2+2k}{k} k^2 2^{k^2+k})$.