

CS 170 Spring 2008 - Solutions to Homework #12

Problem 8.14 (10 points)

We can reduce CLIQUE to the given problem. Given an instance (G, k) of CLIQUE with n vertices, we create G' , which is G together with another n vertices, but without any extra edges. The extra vertices trivially provide an independent set of size $k \leq n$ for any k . Hence, G has a clique of size k if and only if G' has a clique as well as an independent set of size k , since the extra vertices have no edges between them.

Problem 9.5 (15 points total, 5 points each)

- a) Let e be an edge present in T' but not in T . This creates a unique cycle. Also, this cycle must contain at least one edge e' which is not in T' (all the edges in the cycle cannot be from T' since T' is a tree!). Swapping (e, e') gives a tree T_1 which has one more edge from T' as compared to T . We can continue this procedure, until we replace all the edges in T' . Since we always remove edges not in T' , and T' has $|V| - 1$ edges, this takes at most $|V| - 1$ swaps.
- (b) Given T and T' , we include an edge $e = (u, v)$ from T' which is not present in T . This creates a cycle. Now we obtain T_1 by removing one edge from this cycle. However, we need to argue that there exists at least one edge in the cycle which has at least as much weight as e , so that $\text{cost}(T_1) \geq \text{cost}(T_0)$.

To argue this, consider a cut which separates u and v . Then edge (u, v) goes across this cut. Since u and v are on opposite sides of the cut, and the other edges of the cycle form a path between u and v , at least one more edge e' from the cycle must cross the cut. Also, since an edge in the MST must be the lightest edge across a cut separating its endpoints, $w(e') > w(e)$. Thus, we can remove e' . Continuing this argument gives the desired sequence.

- (c) We will now strengthen the previous argument to say that if T is not an MST, then there must be an edge e in an MST T' and an edge e' in T , such that swapping (e, e') strictly *reduces* the cost. Suppose for every edge e in the MST, all the edges in the cycle produced by including e

have weight greater than or equal to e . Then by the previous argument, T contains the lightest edge across every cut, and hence must be an MST. Since we assumed T is not an MST, there must be a cost-reducing swap. This proves that there are no local minima and our algorithm always finds an MST.

To bound the running time, note that after each swap, one edge in the tree is swapped with a lower weight edge. Consider an edge e in the tree. This may be swapped with some edge e_1 , e_1 with e_2 and so on. Since the weights must decrease in this sequence, the length of sequence is at most $|E|$. Also, the tree has $|V| - 1$ edges to begin with. Hence the total number of swaps is $O(|V||E|)$. Also, at each iteration, all possible swaps can be checked in $O(|V||E|)$ time (there are $|V| - 1$ edges in the MST and a DFS on each takes $|E|$ time). Hence the total running time is at most $O(|V|^2|E|^2)$.

Problem 9.6 (10 points)

Let T be the minimum Steiner tree having cost C . Then we can follow the shape of the Steiner tree to obtain a tour of cost $2C$ which passes through all the vertices in the Steiner tree. Let i, j, k be adjacent vertices in the path. Using the triangle inequality, we know that $d_{ik} \leq d_{ij} + d_{jk}$. Hence, we can “bypass” an intermediate vertex j and connect i and k directly if we want, without increasing the cost.

Using this trick, we can bypass all the vertices *not* in V' that are present in the path, and also the vertices of V' which are being visited twice. This gives a path of cost at most $2C$, which passes through all the vertices of V' exactly once. Hence, this path is a spanning tree for V' of cost $2C$. This implies $\text{cost}(MST) \leq 2C$, thus giving the desired approximation guarantee.

Problem 9.8 (15 points total, 5 points each)

- a) Given a formula φ with m clauses as an instance of SAT, it is satisfiable if and only if the maximum number of satisfiable clauses is exactly m . Hence, an algorithm for MAX-SAT easily gives one for SAT.
- b) Let X_j be a random variable obtaining value 1 if the j th clause is satisfied and 0 otherwise. X_j gets a value 0 only when *all* the literals appearing in the clause are simultaneously false in a random assignment, which happens only with probability $\frac{1}{2^{k_j}}$ where k_j is the number literals in the clause. Hence, $E[X_j] = 1 - \frac{1}{2^{k_j}} \geq 1/2$. If X is the number of satisfied clauses in the formula, then $X = \sum_{j=1}^m X_j$ and we have by linearity of expectation

$$E[X] = \sum_{j=1}^m E[X_j] = \sum_{j=1}^m \left(1 - \frac{1}{2^{k_j}}\right) \geq \frac{m}{2}$$

If all clauses contain exactly k literals, then $E[X] = m \left(1 - \frac{1}{2^k}\right)$, which gives an approximation factor of at least $1 / \left(1 - \frac{1}{2^k}\right) = 1 + 1/(2^k - 1)$.

- c) Let N_i denote the number of clauses containing at least one of the variables x_1, \dots, x_i . We claim that after i variables have been assigned, the number of satisfied clauses is greater than $N_i/2$. The statement is true for x_1 by definition of the algorithm. When assigning x_i , we satisfy more than half of *all* the currently unsatisfied clauses containing x_i , which is more than $N_i - N_{i-1}$. Hence, this algorithm satisfies more than half the total number of clauses, achieving an approximation ratio of at least 2.

Problem 9.9 (10 points total, 5 points each)

- a) We first show that any solution S that the algorithm outputs must have the property that $w(S) \geq \frac{1}{2} \sum_{e \in E} w(e)$. Since the size of the maximum cut can at most be $\sum_{e \in E} w(e)$, this guarantees an approximation ratio of 2.

For contradiction suppose that $w(S) < \frac{1}{2} \sum_{e \in E} w(e)$. For each vertex $v \in V$, define $c(v)$ as the total weight of edges incident on v that cross the cut and $w(v)$ as the sum of weights of all the edges incident to v . Then

$$\sum_{v \in V} c(v) = 2w(S) < \sum_{e \in E} w(e) = \frac{1}{2} \sum_{v \in V} w(v)$$

since we count each edge twice, once for each of its endpoints. Thus, there must be at least one vertex v such that $c(v) < \frac{1}{2}w(v)$. If $v \in S$, define S' as $S \setminus \{v\}$, else define S' as $S \cup \{v\}$. All the edges incident to v that were not crossing the cut, will now cross the cut (adding $w(v) - c(v)$), but the ones that were crossing earlier may not (removing $c(v)$), hence

$$w(S') \geq w(S) - c(v) + (w(v) - c(v)) = w(S) + w(v) - 2c(v) > w(S)$$

Hence, there exists an S' , which differs from S only by one element and $w(S') > w(S)$. This means that the algorithm could not have stopped at S , which is a contradiction.

- b) At each iteration, we take $O(n)$ time to find if an S' exists with the required properties. Also, at each step the value of the cut increases and hence the total number of iterations is at most $O(\sum_{e \in E} w(e))$. Thus the running time is $O(n \sum_{e \in E} w(e))$. This is polynomial in the special case when all the edge weights are 1, but not in general.