CMSC 27200 Assignment 9

Rachel Hwang

March 12, 2014

Discussed with James Porter and Andrew Ding.

- **Exercise 8.7**

  First we can show that the 4-dimensional matching problem is NP. Given a proposed solution of n 4-tuples of the form (w_i, x_j, y_k, z_l), we can simply check that there are no repeated values in the set of all w_is, then check that there are no repeated values in the set of all x_js, and the same for y_k, z_l. In other words, confirm at each tuple index $(1, 2, 3, 4)$, accross all n tuples, there are no repeated values. This runs in $O(n^2)$ time since for each value of some tuple[idx], we check that no other tuple has that value at that index, which requires n(n-1)/2 comparisons. We do this for each of 4 indices, so overall $O(n^2)$.

  Now we can show that any instance of the 3-dimensional matching problem can be transformed into the 4-d matching problem. Assume that we have a polynomial time algorithm A for the 4-dimensional matching problem. Given any three disjoint sets W, X, Y, each with cardinality n, and a set T of tuples (w_i, x_j, y_k), we can find a set Z, disjoint from W, X, Y, of n elements. Now for every tuple $(w, x, y) \in T$, we create a tuple $(w, x, y, z) \in T'$ for all $z \in Z$. We can now run our algorithm A on set T'. Any returned solutions of n 4-tuples can then be transformed into a solution of the original 3-d problem simply by excluding the z element of each 4-tuple.

  This transformation is polynomial time, since we add a new set of size n and then for each element in T, do a $O(n)$ iteration.

  Therefore, the 4-dimensional matching problem is NP-complete.

- **Exercise 8.17**

  The zero-sum cycle problem is NP since we can easily verify a solution by walking down the cycle, summing edges as we go and verifying the sum is 0 and that we reach our starting node. Verifying a cycle and summing n edges are both linear, $O(n)$.

  Now, to show that this cycle problem is NP complete, we can show that we can use any polynomial time algorithm which solves it to derive a polynomial time algorithm for the subset sum problem. Assume we have such an algorithm A for the zero-sum cycle problem. We can transform any instance of the subset problem (take a set S of n values) into a input graph G for A. For every $i \in S$, create an edge $e_i \in E$ in G connecting node a_i to b_i, having weight i. This gives us a set of n edges, one for every value in S. We can think of each of these edges as having one *entering* node and one *leaving* node that it connects. We can create a simple cycle for every subset of S by, for each edge e, adding a zero-cost edge (call it a linking edge) between the *leaving* node of e and every *entering* node in the graph. We are now guarenteed to find a simple cycle for every subset because from any e_i corresponding to value i, we can either add another element to the subset we are trying to create by following the edge to the corresponding element, or we can "close" the subset by selecting the edge connecting e_i back to whichever edge this path began with. Thus, one simple cycle for each subset.

  Any zero sum path in the graph must be a zero-sum subset because there are no cycles composed of only linking edges; we only added linking edges which uniquely join a set of weighted edges to a weighted edge. This means that any zero-sum cycles found by A in this graph correspond to a sum-sum subset, the weight of each weighted edge in the cycle corresponding to that value in S.

  The transformation of S to the graph is polynomial. We create each of n edges, then for each of those, a linking edge to every *entering* edge, which is $O(n^2)$. Thus, the zero-sum cycle problem is NP-complete.

- **Exercise 8.22**

  Given A, we can determine whether G contains an independent set of size k as follows:

Add a node s to the graph. Add edges from s to each other node. The resulting graph, G' must be connected, since all noded are neighbors of S, thus it will return a "yes" or "no". There is an independent set of size k in G' if and only if there is one in G (since s cannot be a part of any independent set of any size other than 1). Therefore, A will return the desired solution.

Creating G' is polyonomial, since we add a single edge, constant time, then an edge to each of n nodes, linear time. Thus, we have a polynomial time solution to the Independent Set problem.

- **Exercise 13.1**

Assigning each node a color chosen at random (with each of the three colors assigned with probability of 1/3) will satisfy at least $(2/3)c^*$ edges.

Consider that the expectation of the satisfaction of any given edge is (2/3). This is because for an edge connecting nodes a and b, given that a is assigned some color $c1$, there is a (2/3) probability that b will pick a different color, $p(c2) + p(c3) = 2/3$. By linearity of expectation, to find the expectation of the entire graph, we can simply sum this value over all edges in the graph with n edges, which is $(2/3)n$. This algorithm thus achieves the stated requirement for the expected number of satisfied edges.

The algorithm runs in polynomial time because assuming that picking a color for a node is constant time, we need to do a constant amount of work for each of n nodes, which is $O(n)$.

- **Exercise 13.3**

a. Assume by way of contradiction that this algorithm is not conflict-free. This means that two processes $p1$ and $p2$ have accessed the set at the same time. Our algorithm ensures that to access the set, a proccess's x value must be set to 1. So $x1 = x2 = 1$. However the other stipulation of our algorithm is a proccess can only have entered the set if each of its neighbors are set to zero, implying that niether $p1$ or $p2$ could have entered. This gives a contradiction, hence the algorithm is conflict-free.

Let random variable X_i be a boolean value representing whether process i enters the set. The expectation of X_i is simply $(0.5)(0.5)^d$, the probability that i chooses to set $x_i = 1$ times the probability that each of its neighbors has chosen 0. By linearity of expectation, the expectation over n proccesses will simply be $n \cdot (0.5)^{d+1}$.

b. Intuitively, what will maximize the size of S is, given a proccess i and its d neighbors, for only one of them to choose 1. This means that we want the probability of choosing 1 to be $p(1) = \frac{1}{d+1}$.

We can comfirm this formally. The expression for the expectation of the size of S for n proccesses probability p of 1, is $= n \cdot p(1-p)^d$. We can maximize this function by finding where its derivative over $p = 0$. This works out to be (Wolfram alpha confirms), $\frac{1}{d+1}$.

- **Extra Credit: 13.9**

Strategy: If the seller knows there will be n (even) bidders, let the first $n/2$ go without selling. Now he has some information with which to make his decision. Let x be the highest price offered in the first $n/2$ bidders. From bidder number $n+1$ onward, sell to the first bidder offering more than x. If no such bidder occurs, sell to the last bidder.

Consider the fact that the above strategy works perfectly if x is equal to the second highest bid, meaning that A) the second highest bid occurs in the first $n/2$ bidders and then B) the absolute highest bid does *not*. The probability of A, $p(A) = 1/2$, since it's equally likely that any particular value be in the first half as second half of bidders. $p(B|A) = 1 - \frac{(n/2)-1}{n}$, since we have one less slot to fit the high bid in the first half after finding the second highest there. So $p(B|A) > 1/2$. Therefore, $p(A)p(B|A) > 1/4$. This shows that probability that the above strategy will sell to the highest bid is greater that 1/4.