CS1510 Reduction Problems 6, 10, & 15

Rebecca Negley, Sean Myers

October 26, 2011

11. Show that the Vertex Cover Problem is self-reducible. The decision problem is to take a graph G and an integer k and decide if G has a vertex cover of size k or not. The optimization problem takes a graph G, and returns a smallest vertex cover in G. So you must show that if the decision problem has a polynomial time algorithm then the optimization problem also has a polynomial time algorithm. Recall that a vertex cover is a collection S of vertices with the property that every edge is incident to a vertex in S.

    Solution: The first thing we need to do is get a minimal kth size. To do that, go from 1 to n, n being the number of vertices in the graph. Continually run those values through the deterministic algorithm until the output is 1. That is the minimal vertex cover of graph G

    If we have a graph G, we must remove a vertex l and check to see if the vertex cover is still applicable. A naive approach would be to see if the vertex cover is still of size k. This is a bad idea since removing a vertex will still produce a cover of size k even if l was part of the vertex cover! (unless $k = n$). The thing is, if we remove l, and we can still produce a vertex cover of $k - 1$ with the new graph, then l must be in the vertex cover.

    The algorithmic code for this looks something like this:

```
VCOptimize(G):
    k = 1
    Graph C
    for(i = 1 to n):
        if(VCDecision(G,i) == 1): k = i
    for(Vertex i in G):
        G.remove(i)
        if(VCDecision(G, k-1) == 1): C.add(i)
    return C
```

    All the vertices in C will be the minimal vertex cover in G.

14. Consider the problem where the input is a collection of linear inequalities. For example, the input might look like $3x - 2y \leq 3$ and $2x - 3x \geq 9$ The problem is to determine if there is an integer solution that simultaneously satisfies all the inequalites. Show that this problem is NP-hard using the fact that it is NP-hard to determine if a Boolean formula in conjuctive normal form is satisfiable.

    Solution: We will reduce the CNF satisfiability problem to the linear inequality satisfiability problem. For each clause, we will create one linear inequality. If a variable x appears in the clause, we add that variable to the left side of the inequality. If the negation of some variable x appears in the clause, we add $(1 - x)$ to the left side of the inequality. When we have accounted for every term in the clause, we set the left side > 0. After we have done this for each clause, we create two more inequalities for

each variable x: $x \leq 1$ and $x \geq 0$. Then, we pass our whole set of inequalities into the linear inequality satisfiability problem. If there is an integer solution that simultaneously satisfies all inequalities, then we can give a satisfying assignment to all of the variables in the CNF Boolean formula. If a variable is 1, we can set it to true. If it is 0, we can set it to false. To see that this is valid: Suppose there is some set of integers that satisfies the inequalities but does not satisfy the CNF Boolean formula. Then, there must be some clause where each term is false. This means that, for each term, 0 was added to the left side of the inequality assosicated with that clause. However, this would not have satisfied $LHS > 0$, a contradiction. Suppose there is some satsifiable CNF formula where the associated inequalities had no satisfying set of integers. For the assignment that satisfies the CNF formula, let every term be 1 if true, 0 if false. Then, since each clause is true, at least 1 is added to the LHS of each inequality associated with a clause. Since the last inequalities will always be true if the variables are 0 or 1, every inequality is then satisfied, a contradiction. Hence, the inequalities are satisfiables (with integers) if and only if the CNF formula is satisfiable. Since we run through each term in the CNF formula once and then each variable twice, the CNF satisfiability algorithm will run in polynomial time if the linear inequality satisfiability algorithm runs in polynomial time. Since the CNF formula satisfiability problem is NP-hard, the linear inequality integer satisfiability problem must be NP-hard as well.

16. The input to the three coloring problem is a graph G, and the problem is to decide whether the vertices of G can be colored with three colors such that no pair of adjacent vertices are colored the same color. The input to the four coloring problem is a graph G, and the problem is to decide whether the vertices of G can be colored with four colors such that no pair of adjacent vertices are colored the same color. Show by reduction that if the four coloring problem has a polynomial time algorithm then so does the three coloring problem.

    Solution: We will reduce the three coloring problem to the four coloring problem. For the input graph G, create a graph G' which has all vertices and edges in G and one new vertex v'. For each vertex v in G, create an edge in G' between v and v'. Then, input G' into the four coloring problem. Since v' is connected to every other vertex, it must have its own color. Then, the four coloring problem will give a positive output if and only if the remaining vertices in G' (those that were originally in G) can be colored with three colors such that no pair of adjacent vertices is colored the same color. Hence, we can just return the output of the four coloring problem with input G'. Thus, if the four coloring problem has a polynomial time algorithm, then so does the three coloring problem.