Homework 16

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

7 October 2016

Please provide written or oral feedback on this and all future homeworks. :)

Problem:7

A problem is self-reducible *iff* the optimization form of the problem reduces to the decision form in poly time.

Thus, we show $OptSubsetSum \leq DecisionSubsetSum$

We implement the optimization problem using the decision problem as follows:

Given a mutable collection C of integers $x_1 \dots x_n$ and a goal sum L, first declare a result set, S. For each i from 1 to n, remove x_i from C. If the decision is no longer satisfiable, then add x_i to S and back into C. Return S after reaching all n. Note that if before entering the loop, the decision function returns false, then the empty set should be returned, as this implies no such subset exists.

The logic is as follows: If an element can be removed from the collection and there is still a remaining subset sum, then that element doesn't need to

be in the set for there to be a subset sum since there is still a sum remaining within the elements inside the collection. Conversely, if an element, when removed from the collection, changes the decision from a pass to a fail, then that element must be required in the subset, since a sum existed prior to it's removal but did not exist after it's removal. (thus, the removed element was a member of the subset). The final items in the set are exactly the items needed to construct the sum.

Problem:8

The logic for this solution is extremely similar to the logic for the above solution. In general, this is going to be true of almost all self-reducibility proofs.

We show $OptHamCycle \leq DecisionHamCycle$.

We represent a cycle as a set of edges. The assumption here is that each edge has a pointer to the nodes it connects so that a trace may be easily recreated given the edge set. We implement the optimization problem using the decision problem as follows:

Given a graph G, first declare a result set, S. Should the decision problem return false, then return the empty set since no Hamiltonian cycle exists. For each edge in G, remove the edge from G, creating G'. If G' does not have a Hamiltonian cycle, then add that edge to S and back into G'. Set G to G'.

Similarly to **Problem 7**, if an edge can be removed from the graph and there is still a Hamiltonian cycle present, then that edge is not needed for the Hamiltonian cycle (and may in fact invalidate the cycle by visiting an already visited vertex). By adding to the set an edge if and only if that edge is required for the cycle, you of course wind up with the cycle represented exactly by the set.

Problem:12

a

Determining if a graph $\mathcal{G}$ with n vertices has a clique of size $\frac{3n}{4}$ can be shown to be **NP-Hard** by reducing any problem belonging to the set **NP-Complete** to this problem, which will be called $\frac{3}{4}$ *Clique*. In this case, the chosen problem will be the *k-Clique* problem.

$$k - Clique \leq_p \frac{3}{4}\ Clique$$

For this reduction, the input of *k-Clique* must be transformed into an input for $\frac{3}{4}$ *Clique*. This transformation must consider three different cases:

1. When the clique size given to *k-Clique* is larger than $\frac{3n}{4}$, where n is the number of vertices.

2. When the clique size given to *k-Clique* is less than $\frac{3n}{4}$, where n is the number of vertices.

3. When the clique size given to *k-Clique* is equal to $\frac{3n}{4}$, where n is the number of vertices.

Case 3 is the easiest to handle since the input to *k-Clique* can be given directly to $\frac{3}{4}$ *Clique*. If the output of $\frac{3}{4}$ *Clique* is true, then *k-Clique* also returns true. Otherwise it returns false.

Case 1 and *Case* 2 are more involved; however, they can be handled simultaneously. When the input to *k-Clique* is larger, disjoint vertices can be added to the graph inorder to increase the number of vertices, thereby lowering the ratio. In the case when the input is smaller, the Clique being requested can be grown by 1 by adding a vertex and connecting all vertices

in the original graph to the new vertex. All of these transformations are captured in the following algorithm.

Function: *Clique*

Input: *Graph G, int k*

`/* ` V_G `is the number of vertices in graph G */`

while $k \neq 0.75V_G$ **do**

  if $k > 0.75V_G$ **then**

    `/* argument` G `is modified through function */`

    add_disjoint_vertex(G)

  else

    `/* argument` G `is modified through function */`

    add_connected_vertex(G)

    k++

Return: $\frac{3}{4}$ $Clique(G)$

Two conditions for this reduction must be satisfied: the necessary and sufficient condition on the transformation and the polynomial time bound on the transformation. The necessary and sufficient condition can be proven by construction of graphs. A runtime analysis of the algorithm shows that there are only 2 loops in this algorithm. The outerloop determines if the process has converged and an implicit loop is used to determine how many vertices are in the graph. As a result, this reduction shows $\frac{3}{4}$ $Clique(G)$ is **NP-Hard**.

b

Determining if a graph $\mathcal{G}$ with n vertices has an Independent Set of size $\frac{3n}{4}$ can be shown to be **NP-Hard** by reducing any problem belonging to the set

NP-Complete to this problem, which will be called $\frac{3}{4}\ IS$. In this case, the chosen problem will be the *k-IS* problem.

$$k - IS \leq_p \frac{3}{4}\ IS$$

For this reduction, the input of *k-IS* must be transformed into an input for $\frac{3}{4}\ IS$. This transformation must consider three different cases:

1. When the IS size given to *k-IS* is larger than $\frac{3n}{4}$, where n is the number of vertices.

2. When the IS size given to *k-IS* is less than $\frac{3n}{4}$, where n is the number of vertices.

3. When the IS size given to *k-IS* is equal to $\frac{3n}{4}$, where n is the number of vertices.

Case 3 is the easiest to handle since the input to *k-IS* can be given directly to $\frac{3}{4}\ IS$. If the output of $\frac{3}{4}\ IS$ is true, then *k-IS* also returns true. Otherwise it returns false.

Case 1 and *Case* 2 are more involved; however, they can be handled simultaneously. When the input to *k-IS* is larger, the IS being requested can be lowered by 1 by adding a vertex and connecting all vertices in the original graph to the new vertex. In the case when the input is smaller, disjoint vertices can be added to the graph inorder to increase the number of vertices, thereby increasing the ratio. All of these transformations are captured in the following algorithm.

Function: *Clique*

Input: *Graph G, int k*

`/* ` V_G `is the number of vertices in graph G */`

while $k \neq 0.75V_G$ **do**

  if $k > 0.75V_G$ **then**

    `/* argument` G `is modified through function */`

    add_connected_vertex(G)

    k++

  else

    `/* argument` G `is modified through function */`

    add_disjoint_vertex(G)

Return: $\frac{3}{4}\ IS(G)$

Two conditions for this reduction must be satisfied: the necessary and sufficient condition on the transformation and the polynomial time bound on the transformation. The necessary and sufficient condition can be proven by construction of graphs. A runtime analysis of the algorithm shows that there are only 2 loops in this algorithm. The outerloop determines if the process has converged and an implicit loop is used to determine how many vertices are in the graph. As a result, this reduction shows $\frac{3}{4}\ IS(G)$ is **NP-Hard**.

c

In order to prove that problem (c) is NP-Hard, we need to reduce an NP-Complete problem to (c). We will call (c) *Algo*, where the input to *Algo* is a graph H and an integer j. The problem that will be reduced to *Algo* is Independent Set, in this problem we will use the abbreviation IS.

The basic idea of this reduction is to transform the input of IS, a graph

G and an integer k, to the input for *Algo* and then return the output from *Algo*.
The reduction is as follows:

Algorithm *IS(graph G, integer k)*
$H = G +$ a Clique of size k, where each vertex in the Clique is connected to each vertex in the graph G
$j = k$
return $Algo(H, j)$

---

d

e

f