Homework 17

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

17 October 2016

Please provide feedback on this and all future homeworks :)

Problem 9:

The optimization version of the clique problem with returns the large clique in undirected graph $\mathcal{G}$ can be shown to be self reducible in polynomial time if the decision version of the clique problem has a polynomial time algorith. Symbollically, this means the following.

$$Clique_{opt} \leq_{poly} Clique_{eq}$$

Since the optimization version of the clique problem returns the largest clique within the graph $\mathcal{G}$, the following algorithm demonstrates the polynomial time reduction under the assumption of a polynomial time algorithm for the decision problem.

Function: $Clique_{opt}$
Input: $Graph\ \mathcal{G}$

```
/* n is the number of vertices in graph G                    */
/* this loop will eventually exit via return.  All graphs
   have a 1 clique                                            */
while n ≠ 0 do
    if Clique_dec(G, n) then
        while Vert_G ≥ n do
            v = Remove_Vertex(G, n)
            if Clique_dec(G, n) then
                continue
            else
                mark_vertex(v)
                Read_VertextG, v
        Return: G_verts
    else
        n − −
```

Problem 10:

The vertex cover problem is self-reducible *iff* the optimization form reduces to the decision form. Let *OptVC* represent the optimization form, and *DecVC* represent the decision form. Our algorithm operates in $\mathcal{O}(n)$ times the time of *DecVC*.

In order to find the *smallest* vertex cover, we first for a vertex cover of size 1, then size 2, up until the decision problem returns true. Note that this can take up to n searches, and that this will always terminate because all graphs have a cover of size n. Once we've identified the size of the smallest vertex cover, which we call k, we then iterate through each vertex in the graph, removing the vertex if it's presence does not have an impact on the decision solution. If it does have an impact on the decision solution (*i.e.* removing it causes *DecVC* to return false), then it must me a member the smallest remaining vertex cover since removing it caused there to be no remaining smallest vertex cover. We return all remaining vertices.

```
Function: OptVC
Input: Graph 𝒢
Let k := 0 while true do
    if DecVC(G, k) then
        break;
    k++
Let S be an empty set
foreach v ∈ G do
    G′ = G − v
    if not DecVC(G′, k) then
        S = S ∪ v
        G′ = G
    G = G
Return: S
```

Problem 13:

We call the problem of finding a satisfying assignment of values to linear variables *SMT*. While the literature usually refers to *SMT* as a slightly more general invariant, for our purposes this name is descriptive enough.

To show that *CNF-SAT* $\leq_{poly}$ *SMT*, we translate each of the conjunctive clauses individually into linear inequalities. We provide a 1 to 1 mapping from propositional logic formulas to linear inequalities that maintains correctness (that is, the formula is satisfiable *iff* the inequality is satisfiable). The correctness invariant should be fairly obvious.

Iterate through the propositional logic conjunctions. For each formula $x_1 \vee x_2 \vee \ldots x_n$, convert each singleton literal x_i into the inequality $x_i > 0$ if the literal is not negated and $x_i \leq 0$ if the literal is negated. This encodes each of the literals as a negative number if negated, and a non-negative number if not negated. For each disjunction in each conjunctive clause, encode the disjunction $a \vee b$ as $a + b > 0$, where each singleton literal a is converted as above. Obviously, this inequality is satisfiable *iff* the proposition is satisfiable due to the properties of order fields (transitivity of addition).

Problem 14:

It can be shown by reduction that if there exists a polynomial time algorithm for the *4-coloring* problem there also exists a polynomial time algorithm *3-coloring*. Mathematically this relationship can be expressed as follows.

$$3coloring \leq_{poly} 4coloring$$

The following algorithm demonstrates this reduction.

Function: $3 - Coloring$
Input: *Graph* $\mathcal{G}$

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

$\mathcal{G}' = add\_connected\_vertex(\mathcal{G})$
Return: $4 - coloring(\mathcal{G}')$

The reason this algorithm works is because if a graph had a *3-coloring*, the addition of a vertex connected to all vertices forces the addition of a new color because if any of the three original colors are given to the new vertex, it will create a violation of the coloring. As a result, the new graph will have a *4-coloring*. In a similar manner, the addition of a new vertex to a graph that did not originally have a *3-coloring* will not result in a *4-coloring*.

Lastly, the addition of a new vertex and the subsequent connecting of all the original vertices to the new vertex can be done in polynomial time. This is apparent if the original graph was stored in an adjacency matrix and a new row and column are added.