Homework 19

Robbie McKinstry, Jack McQuown, Cyrus Ramavarapu

17 October 2016

Problem 17:

The Fixed Hamiltonian path problem determines if an undirected graph $\mathcal{G}$ has a simple path that spans all vertices in $\mathcal{G}$ between two given vertices x and y. If there exists a polynomial time algorithm for this problem, it can be shown by reduction, that there exists a polynomial time algorithm for the Hamiltonian Cycle problem.

The Hamiltonian cycle problem takes an undirected graph $\mathcal{H}$ and determines if there is a simple cycle spanning all the vertices of $\mathcal{H}$. It is obvious that if the graph contains a vertex with a neighborhood size ≤ 1, there cannot be a cycle because this vertex will either represent a deadend or an unreachable vertex. Accounting for this problem and iff it is assumed that the Fixed Hamiltonian path problem can be solved in polynomial time, this problem can also be solved in polynomial time as demonstrated by the following algorithm.

```
Function: $HamCycle$
Input: $\mathcal{G}$
for $Vertex\ v\ in \mathcal{G}$ do
    if $Neighborhood(v, \mathcal{G} \leq 1$ then
        Return: 0
for $Vertex\ v\ in\ \mathcal{G}$ do
    /* Neighborhood(Vertex v, Graph G)                          */
    /* Returns the set of vertices adjacent to v in graph G
       */
    for $Vertex\ u\ inNeighborhood(v, \mathcal{G})$ do
        if $FixedHamPath(v, u, \mathcal{G}) == 1$ then
            Return: 1
Return: 0
```

Since it is assumed that the *FixedHamPath* algorithm runs in polynomial time, this algorithm also runs in polynomial time because it only through all the vertices in the outerloop and in the innerloop will loop through a maximum of $n - 1$ vertices. As a result, the *FixedHamPath* problem will be called $\mathcal{O}(n^2)$ times. Therefore this algorithm will run in Poly-time.

Although the above algorithm demonstrates the reduction of the Fixed Hamiltonian path problem from the Hamiltonian cycle problem, the repeat calls to *FixedHamPath* are inelegant. The reduction can be performed with a single call to *FixedHamPath* by transforming the input graph $\mathcal{G}$ in such a way that there will only be a Fixed Hamiltonian Path if and only if there is also a Hamiltonian Cycle. This can be accomplished by adding two disjoint vertices to the original graph named a and b. Both a and b are then made adjacent to two members in the neighborhood of an arbitrary node, x, in the original graph $\mathcal{G}$. The new graph, $\mathcal{H}$ will only have a Fixed Hamiltonian Path between a and b if there is a cycle that spans all the nodes in the original graph. This arises from the lack of a 'start point' in the cycle and the need to bridge x. Pictorially this is represented by the following sketch:

Algorithmically, this is covered by the following code once again assuming that there exists a polynomial time algorithm for the Fixed Hamiltonian path problem and the abscence of any vertices with a neighborhood ≤ 1.

```
Function: $HamCycle$
Input: $\mathcal{G}$
begin
    $\mathcal{H} = add\_disjoint\_vertex(a, \mathcal{G})$
    if $Neighborhood(v, \mathcal{G} \leq 1$ then
        Return: 0
    $\mathcal{H} = add\_disjoint\_vertex(b, \mathcal{H})$
    /* $u$ is an arbitrary vertex in $\mathcal{H}$                        */
    /* $u + 1$ is a member of neighborhood(u) in $\mathcal{H}$            */
    /* $u - 1$ is a member of neighborhood(u)$\neq$ u+1 in $\mathcal{H}$  */
    $\mathcal{H} = create\_edge(a, u + 1, \mathcal{H})$
    $\mathcal{H} = create\_edge(b, u - 1, \mathcal{H})$
    Return: $FixedHamPath(a, b, \mathcal{H})$
```

Due to the lack of explicit loops in this algorithm, it is obvious it runs in polynomial time under the aforementioned assumptions. However, for the sake of clarity, vertices and edges can be added in Poly-time depending on the datastructure used to store the graph.

Problem 18:

Personally, we found that the Hint got us 90% of the way through this problem, and that last 10% did not require much creativity. The Hint did all of the heavy lifting.

Given a graph H and an integer ℓ, we seek to solve vertex cover using an algorithm for the problem described, which we call *MinimumConnection* or

MinCon for short. We show that vertex cover can be solved in poly time if *MinCon* can be solved in polytime. Below is an high-level description of our algorithm.

First, let G be a graph with a single vertex, *Center*. Label each vertex in H uniquely, and for each vertex in H add a vertex in G and connect that vertex to C, maintaining the label from H in G. This constructs G such that a path exists from any vertex to another by passing through C. Now, for each edge e in H, add a vertex v to G. Connect v to another node v' if and only if e is an edge of the vertex with the label v' in H. (Recall that the labels were copied over, so a vertex v' in G has a correspondingly vertex in H.) This results in each vertex added in this manner possessing two edges (since every edge in H connects exactly two vertices). These edges are also connected to each other by pathing through *Center*.

Let *Nodes* be the set of vertices in G labeled as corresponding to a vertex in H (and thus not an edge in H). Let *count(edges)* be the number of edges in H. Finally, let $R = \{x \in G\} - Nodes$. Return the result of *MinCon(G, R, 1+* $\ell +$ *count(edges))*

MinCon is satisfiable *iff* there exists a vertex cover of size ℓ on H. An algorithm for *MinCon* must add to U all of the nodes in R, which is *Center* and all of the vertices in G representing edges in H (that is, all of the vertices in G not representing vertices in H). This leaves ℓ vertices left to be added to the set U, and only the vertices left in *Nodes* to select from. Selecting a vertex v from *Nodes* creates a path connecting all of v's edges to *Center* since v is connected to *Center*, so all of the vertices in R will be connected to each other if at least one of the vertices in *Nodes* and also in U has an edge to that vertex.

This algorithm operates in polynomial time (iff *MinCon* operates in polynomial time).