Homework 22

Brian Knotten, Brett Schreiber, Brian Falkenstein

October 18, 2018

8

HamiltonianCycle $\leq_p$ DoubleFixedHamiltonianPath

HamiltonianCycleAlgorithm(G):
    return $\bigvee_{v \in G}$ DoubleFixedHamiltonianPathAlgorithm(G, v, v)

A Hamiltonian Cycle is a special case of a Hamiltonian Path where the start and end vertices happen to be the same vertex. This algorithm takes polynomial time because it makes at most n calls to the (assumed polynomial) DoubleFixedHamiltonianPathAlgorithm.

SingleFixedHamiltonianPath $\leq_p$ DoubleFixedHamiltonianPath

SingleFixedHamiltonianPath(G, u):
    return $\bigvee_{v \in G}$ DoubleFixedHamiltonianPathAlgorithm(G, u, v)

A Single Fixed Hamiltonian Path can be discovered with a Double Fixed Hamiltonian Path algorithm by trying all possible endpoint vertices and seeing if a hamiltonian path exists between those two vertices. This algorithm takes polynomial time because it makes at most n calls to the (assumed polynomial) DoubleFixedHamiltonian-PathAlgorithm.

DoubleFixedHamiltonianPath $\leq_p$ HamiltonianCycle

DoubleFixedHamiltonianPathAlgorithm(G, u, v):
    return $\bigvee_{(u',v') \notin G} HamiltonianCycleAlgorithm(G + (u', v'))$

All Hamiltonian Cycles are simple Hamiltonian Paths with an extra edge from the start vertex to the end vertex. Therefore, if there exists a Hamiltonian Cycle in the graph G' which contains an extra edge from the start vertex to the end vertex, then there also contains a Hamiltonian Path in G without the extra edge. This algorithm takes polynomial time because it makes at most n calls to the (assumed polynomial) HamiltonianCycleAlgorithm..

10

Define $HCD(G)$ to be the algorithm for the decision problem for if a Hamiltonian Cycle exists for a graph G, and $HCO(G)$ to be optimization problem for actually finding the Hamiltonian Cycle in graph G. That is $HCD(G)$ will output 1 if an HC exists in G, and a 0 if not, and $HCO(G)$ will actually output the edges that constitute a HC in G, or 0 if one doesn't exist. The claim is that $HCO(G) \leq_p HCD(G)$, IE Hamiltonian Cycle is self reducible.
In order to prove this, we must show that we can use $HCD(G)$ to output a list of edges constituting a HC in G, in polynomial time.
First, assume graph G is defined as a list of vertices and a list of edges. Consider the following pseudo-code:

```
HCO(V, E):
      if HCD(V, E):                                        #initial check, make sure G has an HC
           HC = []                                         #initialize solution
           while E.hasNext:                               #continue until no edges left
                testEdge = E.pop                          #remove an edge from the graph
```

```
            if HCD(V, E):                           #check if HC exists in G minus one edge
                  HC.append(testEdge)              #if so, add the edge we removed to solution
        return isHC(HC, V, E)                        #function to determine if a path is a HC for a graph
    return 0
```

The general strategy of this algorithm is to look at an edge e in G, determine if we can still form an HC in G when we remove e. If so, we can safely add e to our solution. If not, we can exclude e. We repeat this until there are no edges left in G, and then we test if the cycle we've found is actually a HC. Note that $isHC$ could be defined very simply by checking that:

- All edges in HC exist in E
- No vertex in V is visited more than once in HC
- HC spans all vertices in V

The number of times HCD will be called inside of HCO is at most n, where n is the number of edges in G, as each time it is called at least 1 edge is removed. Similarly, $isHC$ will take at most n time, as if HC is a hamiltonian cycle, the max edges it could contain will be n. This results in a total run time of $n + n = O(n)$, a polynomial. Thus, we have proven that Hamiltonian Cycle is self reducible, and if we can determine whether a graph has an HC in polynomial time, we can find the HC in polynomial time.

12

Vertex Cover is self-reducible if Optimal Vertex Cover $\leq_p$ Vertex Cover Decision. An algorithm for Optimal Vertex Cover takes as input a graph G and returns k vertices where k is the smallest number of vertices needed for a vertex cover.

OptimalVertexCoverAlgorithm(G):
First, find the minimum number of vertices needed for a vertex cover by continually incrementing the number of vertices allowed until a vertex cover possible. Let $k = 0$

    while !VertexCoverDecision(G, k):
        $k = k + 1$

    for each $v \in G$:
        # Try removing v and all edges adjacent to v in the graph. That is, assume v is in a solution to the Vertex Cover.
        if VertexCoverDecisionAlgorithm($G - v, k - 1$): # If a vertex cover is possible with the rest of the graph,
            $S = S \cup \{v\}$ # Then v was a viable vertex to cover in the optimal solution, so append it to the solution set.
            # Continue with the reduced problem size
            $G = G - v$
            $k = k - 1$

    Return S

The number of times VertexCoverDecisionAlgorithm will be called inside OptimalVertexCoverAlgorithm will be at most $2n$, where $2n$ is the number of vertices in G. Thus, if VertexCoverDecisionAlgorithm has a polynomial time algorithm, then so does OptimalVertexCoverAlgorithm, since $2n$ calls to a poly-time algorithm is still poly-time. So Vertex Cover is self reducible.