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 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.

DoubleFixedHamiltonianPath $\leq_p$ SingleFixedHamiltonianPath

DoubleFixedHamiltonianPathAlgorithm(G, u, v):
    Let $nexts = \{v\}$
    while $nexts \neq \phi$
        Let $next = nexts.pop()$
        if SingleFixedHamitonianPathAlgorithm($G - next, u$) $\wedge nexts = \phi$:
            return False
        $G = G - next$
        $nexts = nexts \cup next.unvisitedNeighbors$ except u
    return True

If there exists a Hamiltonian path in G from u to v, then there must exist a Hamiltonian path in $G - v$ from u to one of v's neighbors. Otherwise, there was never a Hamiltonian path in G from u to v to begin with. This algorithm visits each of v's neighbors, and neighbors of neighbors, and so on, and if all vertices are visited without a

Hamiltonian path being deemed impossible, then it can return true. Otherwise, at some point a Hamiltonian path from u to v is impossible, so return false. This algorithm takes polynomial time because it makes at most n calls to the (assumed polynomial) SingleFixedHamiltonianPathAlgorithm.

Summary

If a Double Fixed Hamiltonian Path algorithm is poly-time, then it is possible to make both a poly-time Hamiltonian Cycle algorithm and a poly-time Single Fixed Hamiltonian Path algorithm.
If a Single Fixed Hamiltonian Path algorithm is poly-time, then it can make a poly-time Double Fixed Hamiltonian Path algorithm, which in turn can make a poly-time Hamiltonian Cycle algorithm.
If a Hamiltonian Cycle algorithm is poly-time, then it can make a poly-time Double Fixed Hamiltonian Path Algorithm, which in turn turn can make a poly-time Single Fixed Hamiltonian Path.

10

In order to prove Hamiltonian Cycle is self reducible, we must show that we can use HamiltonianCycleDecisionAlgorithm to output a list of edges constituting a HC in G, in polynomial time.

OptimalHamiltonianCycle(G):
    if not HamiltonianCycleDecision(G):
        return False

    for each edge $e \in G$:
        if not HamiltonianCycleDecisionAlgorithm($G - e$):
            $path = path + e$
        else:
            $G = G - e$
    Return $path$

The general strategy of this algorithm is to look at an edge e in G and determine if we can still form a Hamiltonian Cycle 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. The number of times HamiltonianCycleDecisionAlgorithm will be called inside of OptimalHamiltonianCycle is at most n, where n is the number of edges in G. Thus, we have proven that Hamiltonian Cycle is self reducible, since if we can determine whether a graph has a Hamiltonian Cycle in polynomial time, then we can find the Hamiltonian Cycle 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.