CS 577: Introduction to Algorithms 11/30/2015

Homework 10 Solutions (Review Problems)

Instructor: Dieter van Melkebeek TA: Bryce Sandlund

Problem 1

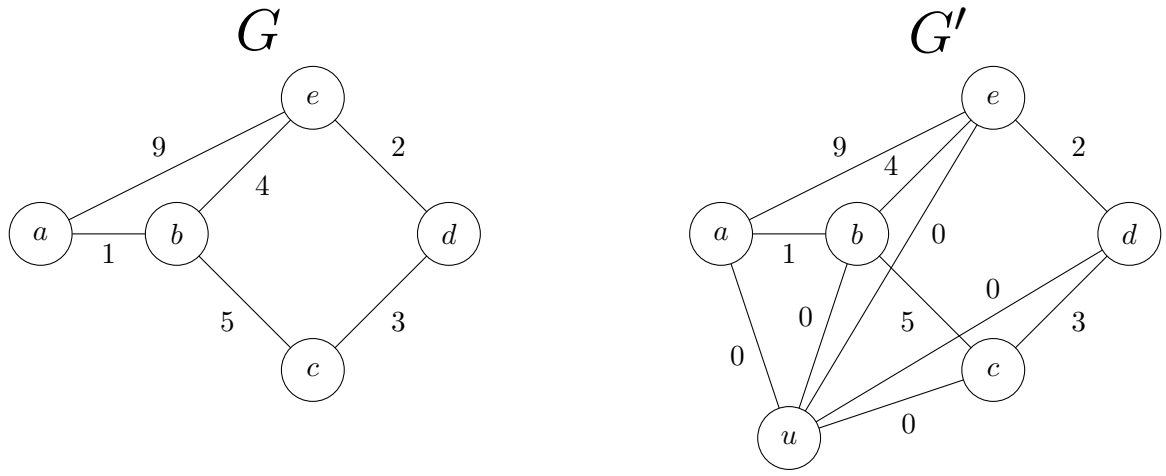
Recall that the decision version of the satisfiability problem for boolean formulas is as follows: Given a boolean formula written as ANDs, ORs, NOTs, and parenthesis of variables $x_1, \ldots, x_n$, output 'yes' if there exists an assignment of `true` or `false` to each variable such that the formula evaluates to `true`, otherwise output 'no'. The corresponding search problem then asks for a satisfying assignment of each variable or an indication that no such satisfying assignment exists.

We can reduce search to decision using the following strategy. First, we ask if a satisfying assignment exists. If not, we output that no satisfying assignment exists and we are done. Otherwise, assign the value `true` to variable x_1 and create a new formula by replacing x_1 with `true` and simplifying. Then check if a satisfying assignment exists in this new formula. If yes, then x_1 can be assigned the value `true`. If not, then x_1 must have the value `false`, since a satisfying assignment of the original formula existed, and x_1 can only take the values `true` or `false`. Therefore we can replace x_1 with `false` in the original formula and simplify. In either case, we have found an assignment of x_1 and produced a new formula such that any assignment of the variables $x_2, \ldots, x_n$ in the new formula correspond to a valid assignment in the original formula. Therefore, we can recurse on the new formula to determine a valid assignment of the remaining variables. In this way, we can solve the search problem for satisfiability of boolean formulas by making polynomially-many ($O(n)$) calls to decision and doing polynomial-time amount of extra work.

Problem 2

Call the version of the Traveling Salesman Problem (TSP) where the tour does not need to end in the same city as it starts Path-TSP. In the same way, call the original formulation where the tour does need to start and end in the same city Circuit-TSP. Denote the optimization version of these problems Path-TSP-optimization and Circuit-TSP-optimization, respectively. Recall that for the optimization version of TSP, we must find the value AND path of the shortest route that visits each city exactly once. Furthermore, to show an optimization problem A polynomial-time mapping-reduces to an optimization problem B, we must show that for any instance x_A of problem A, we can map it to an instance x_B of B such that we can construct the optimal solution to x_A given the optimal solution to x_B. More formally, we must find a function g that maps an instance x_A of problem A to an "equivalent" instance x_B of problem B, and another function h that takes an instance x_A of A and a solution y_B of x_B and maps to a solution y_A of x_A. The solution y_A of x_A should be optimal, and all mapping and construction must be computable in polynomial time. Finally, optimization problems A and B are equivalent under polynomial-time mapping reductions if A polynomial-time mapping-reduces to B and B polynomial-time mapping-reduces to A. We will start by showing Path-TSP-optimization polynomial-time mapping-reduces to Circuit-TSP-optimization.

Figure 1: Path-TSP-optimization $\leq$ Circuit-TSP-optimization Example



Path-TSP-optimization $\leq$ Circuit-TSP-optimization

Given an instance of Path-TSP with graph G, construct a copy of graph G, G'. Then, add an additional vertex u and edges of weight 0 from u to all vertices in G'. Thus we have $g(G) = G'$. Now, run Circuit-TSP on the new graph G', which returns a TSP circuit C of G' of minimal cost, or states that no such circuit exists. If no circuit exists, return that no TSP path exists in G; otherwise, construct a path P in G by removing vertex u from C and its corresponding edges. Therefore, $h(G, C) = P$. We claim that P is a minimal-cost path in G or no path exists in G that visits each vertex exactly once. We will show this by showing an equivalence of TSP paths in G and TSP circuits in G'.

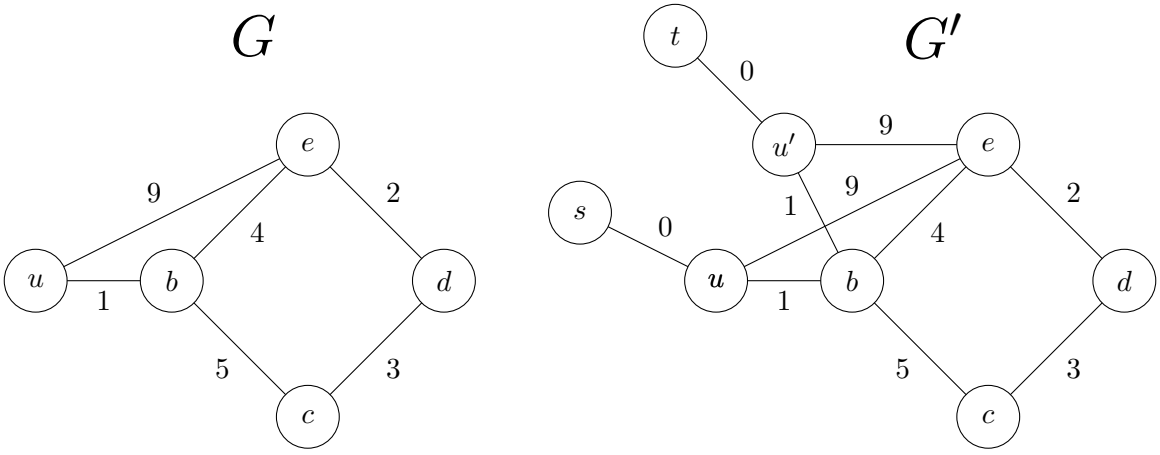
Note that in the method used to construct P, we removed edges of cost 0 from C, so P has the same cost as the circuit C. Further note that since C visits every vertex of G' exactly once, and P visits every vertex C visits other than u, which does not exist in G, P visits every vertex of G exactly once. Finally, since this construction did not depend on C being minimal, this shows for every TSP circuit in G' we can construct a TSP path in G of equivalent cost.

Now, in the other direction, note that for any TSP path X in G, starting at some vertex s and ending at some vertex t, we can construct a TSP circuit Y in G' of equivalent cost by adding the edges (t, u) and (u, s), both of cost 0, to Y. This equivalence shows that if no TSP circuit exists in G', then no TSP path exists in G, otherwise, the minimal cost TSP circuit in G' is the same cost as the minimal cost TSP path in G, completing the reduction. Finally, since constructing the graph G' and removing u from the minimal cost circuit C in G' can be done in linear time, the reduction can be computed in polynomial time.

Circuit-TSP-optimization $\leq$ Path-TSP-optimization

Given an instance of Circuit-TSP with graph G, construct a new graph G' by performing the following steps. First, let $G' = G$. Then, arbitrarily choose a vertex u of G' and duplicate it, copying any edges incident to u to the duplicated vertex u'. Now, create auxiliary vertices s and t and connect s to u and t to u', both with edges of cost 0. So we have $g(G) = G'$. Run Path-TSP on the new graph G', which returns a TSP path P of G' of minimal cost or states that no such

Figure 2: Circuit-TSP-optimization $\leq$ Path-TSP-optimization Example



path exists. If no path exists, return that no TSP circuit exists in G; otherwise, note that since auxiliary vertices s and t have degree 1, if any TSP path exists in G', it must visit s and t first and last. Therefore, we can construct a circuit C in G from P by removing vertices s, u, t and their corresponding edges and adding the edge (z, u), where z is the vertex immediately preceding u' in P. So $h(G, P) = C$. We claim that C is a minimal-cost circuit in G or no circuit exists in G that visits each vertex exactly once. Again, we will show this by showing an equivalence of TSP circuits in G and TSP paths in G'.

Since the edges (s, u) and (t, u'), removed from P to construct C, are of cost 0, and the edge (z, u) is the same cost as edge (z, u'), it follows that C has the same cost as P. Furthermore, since P visits every vertex in G', and C visits every vertex that P visits except for u', t, and s, it also follows that C visits every vertex in G. Then, again, since the construction of C did not depend on the minimality of P, this shows that for any TSP path in G' we can construct a TSP circuit in G of equivalent cost.

In the other direction, consider a TSP circuit X in G. Construct a TSP path Y in G' by removing the edge (z, u), where z is a neighbor of u in X, and adding edges (z, u'), (u', t), and (u, s). Again, since (u', t) and (u, s) are of cost 0, and (z, u') is the same cost as (z, u), Y has the same cost as X. Therefore if no TSP path exists in G', then no TSP circuit exists in G, otherwise, the minimal TSP path in G' is the same cost as the minimal TSP circuit in G. Since creating the graph G' and the circuit C can be done in linear time, the reduction can be computed in polynomial time and so we are done.