Elijah Chou

Dr. Michelangelo Grigni

CS 326

4/4/2022

HW 5

Diagram

Description automatically generated

The simplest way of transforming the graph would be to replace every weighted vertex with a vertex-edge-vertex pair (as shown above). Since we are replacing every vertex with one edge and two vertices, we would have |V| more edges and twice the number of vertices (with two additional ones for sink and source). Therefore, there are |E| + |V| = |E’| and 2\*|V| + 2 vertices, which could simplify to 2\*|V|. These are for the new G’ graph.

The source and sink of the graph would be the two new ones that connect to all of the sources or all of the sinks in the original graph with infinite edge capacities (labeled s and t on the graph on the right).

1. Since we know that after constructing the flow network G’, we should start off with a graph that should have capacity restraints that are all equal to 1, which are integers. Knowing this, this means that when deciding the δ = min(cf(u,v), ef(u)) for a valid push, it will always choose the cf(uv) as the minimum since the capacity restraints of 1 should always be less than or equal to the original values of the edges in the bipartite graph G. This means that the excesses will always decrease by an integer value and thus the preflow will stay integer valued.  
     
   Every push forward or backwards should also be a saturating push because all capacity constraints are equal to 1. A saturating push is described as a push that causes the cf(u,v) = 0, and this should be true of all pushes that occur because in all pushes in this situation δ = cf(u,v) = 1, which ultimately “saturates” the capacity on the flow network G’.

Finally, now that we know that all pushes will be a saturating push, we know that the generic push-relabel algorithm should solve this within O(VE) time because, as shown by Lemma 26.22 on the bound of saturating pushes and Corollary 26.21 on the bound of relabel operations, the number of operations the algorithm will take is O(VE) and O(V^2), respectively.

To show that this simplifies to O(VE) total time. We can show that the O(V^2) relabel operations happens in O(VE) time because the taking of the minimum of things in the relabeling function takes time proportional to the out degree of each vertex. And since each vertex is relabeled once, the time it takes should be |V| \* ∑v∈V (out-degrees of v). Since the sum of all the out degrees should be equal to the number of edges, the expression simplifies to |V||E|, which simplifies to O(VE) time.

Saturating pushes and relabeling operations take O(VE) time each, which means that the generic algorithm should solve the problem in O(VE) time.

1. To prove this via contradiction, we need to find some minimum cut for which a vertex with v.h > k is on the sink side of that cut. There should be a residual flow network in which that minimum cut is saturated. Any other vertices that were also on the sink side of the cut that has an edge going to v in this residual flow network should be greater than k. This is because since it’s h value cannot be equal to k and it could be only at most one less than v. With this method, let S be the set of vertices on the sink side of the graph which have an h value greater than k.

Suppose that there were some simple path from a vertex in S to s. We know that there is no path in the res. flow network from a S vertex to s. This is because the path cannot get from above k to 0 without going through k, which also means that the height could only decrease by at most 1. As such, there must be a minimum cut for which S lies entirely on the source side of the cut, and this is because a minimum cut corresponds to disconnected parts of the graph for a maximum flow. This is further bolstered by the fact that there is no path from the S vertices to s, and all of this contradicts how v is selected.

To prove that a height function still exists despite the contradiction, suppose we had an edge from u to v in the graph. We knew from before that u, h ≤ v.h + 1, but this also means that if u.h > k, so must be v.h. Therefore, if both were above k, we would be making them equal and the height function would still be valid. Also, if just v.k were above k, then we have not decreased it’s h value, meaning that the inequality also still must hold. Since the value of s.h, and t.h were not changed, all the required properties to have a height function after modifying the h values are fulfilled.

1. The first step is to compute the strongly connected components of the directed graph and look at the resulting component graph. The component graph should be acyclic will have at most the original number of vertices and at most the original number of edges. The acyclicity of the graph makes it eligible for the transitive closure algorithm.   
     
   For every edge from S1 to S2 in the transitive closure of the component graph, we add an edge from every vertex in S1 to one vertex in S2. The time it takes to do this is O(V + E\*). To add this to the final total, the time it takes should be less than or equal to f(|V| + |E|) + O(V + E\*).
2. Suppose we have a graph G that has vertices x, y, and z with an edge between y and z with a weight of 1. Let the vertex a be the sink. In this hypothetical scenario, w^(y, z) cannot be defined because the heights of both y and z are infinite. If G is also strongly connected and there are no negative weight cycles in it, then the height function should be defined well for any choice of source vertex because every vertex is reachable from every other.

Then, we need to check if the new weight function w^ satisfies the two properties.

1. Let p = <u = v0, v1, …, vk=v> be a shortest path from u to v using weight function w. Then:

Since h(u) and h(v) are independent of p, w must minimize w(p) iff w^ minimizes w^(p).

1. Triangle inequality says that for any vertices v, u ∈ V, we have

δ(v, u) ≤ δ(v, z)+w(z, u)

Thus, if h(u) = δ(v, u) then we have

w^(z, u) = w(z, u) + h(z) − h(u) ≥ 0