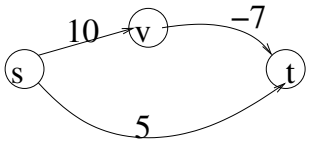
CS 6901 (Applied Algorithms) – Lecture 8

Antonina Kolokolova

October 13, 2016

1 Single-source shortest path with negative weights/cycles: Bellman-Ford algorithm

Dijkstra's algorithm is fast and powerful when the weights of the edges are arbitrary positive numbers. However, it breaks down when edges are allowed to be negative numbers. For example, consider a graph with vertex s connected by a vertex t by a directed edge of length 5, and also by a path starting with an edge (s, v) of weight 10, followed by an edge (v, t) of weight -7. Then, the distance from s to t should be 3, via the path s, v, t. However, Dijkstra's algorithm will take t off the queue first when its distance value is still 5, and never update it again.



Another issue that comes up when negative weights are introduced is the possibility of negative cycles, that is, cycles of total weight < 0. In that case, the shortest (minimal weight) path is meaningless: the minimal weight is achieved by going to the negative cycle, and circling around it forever. Thus, an algorithm is needed that would be able to detect negative cycles, and if there are none, compute the shortest distances. The following algorithm, due to Bellman and Ford, solves this problem in time $O(nm)$.

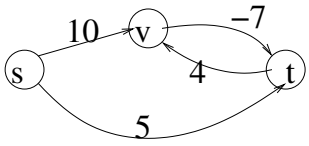
Bellman-Ford(G,c)
Initialize $s.key = 0$, $s.pred = null$
$\forall v \neq s, v.d = \infty; v.pred = null$
for $i = 1$ **to** $n - 1$
  for each edge (u, v)
    if $u.d + c(u, v) < v.d$ **then**
      $v.d = u.d + c(u, v); v.pred = u$
for each edge (u, v)
  if $u.d + c(u, v) < v.d$ **then**
    output "Negative cycle found"
return G

Now, running this algorithm on the graph above, we will have the following distances after the corresponding number of iterations of the first **for** loop (assuming order (v, t), (s, t), (s, v)):

	s	v	t
0	0	∞	∞
1	0	10	5
2	0	10	3

Alternatively, consider adding the edge of weight 4 from t to v. Now, the table changes as follows (assuming order (v, t),(s, t),(s, v),(t, v)). And during the check for a negative cycle, $t.d > v.d + c(v, t)$.

	s	v	t
0	0	∞	∞
1	0	9	5
2	0	6	2



To show that Bellman-Ford algorithm is correct, we need to prove two properties. First, that if there are no

negative cycles, then it correctly computes the distances. And second, if there is a negative cycle, then the final **for** loop catches it, but the check passes if there are no negative cycles.

To prove that the algorithm correctly computes shortest distances when there are no negative cycle, consider a shortest path from s to an arbitrary vertex t: since there are no negative cycles, such a path exists and it is a simple path. Moreover, for any vertex v on this path, its shortest distance from s is the distance it has on this path. Now, we claim that at the first iteration of Bellman-Ford, all vertices that have shortest distance from s by their edge from s will have the correct distances computed. Similarly, on i^{th} iteration of the outer **for** loop, all vertices that are i edges from s on their shortest path will be computed correctly (by induction). Therefore, since no simple path in an n-vertex graph can have more than $n-1$ edges, after $n-1$ iterations all shortest paths will be computed correctly.

Now, suppose that there are no negative cycles. Then, since by the previous statement, the nested **for** loops computed the shortest paths from s to every vertex v, it holds that $v.d \leq u.d + w(u,v)$ (otherwise $v.d$ would not have been the shortest distance to v). Thus, for every edge the check in the final **for** loop passes. Alternatively, suppose there is a negative cycle in the graph. Then for at least one vertex on the cycle its $v.d$ can be improved by coming from the previous vertex in the cycle: to see that, just sum the distances to vertices in the cycle, and notice that this sum consists of the sum of $v.d$'s in the cycle plus the sum of the edges in the cycle; the latter sum is less than 0.

As an application, consider the following money-making idea. Suppose that 1 USD is worth 1.3 CAD, 1 CAD is 0.7 Euro and 1 Euro is 1.2 USD. Then by exchanging 1 USD you would get 1.3 CAD; exchanging 1.3 CAD to Euro gives you 0.91 Euro, and exchanging 0.91 Euro to USD gets you 1.10 USD, which is more than you have started with. However, if instead 1 CAD would convert to 0.6 Euro, you would lose money. Thus, the cycle of exchanges that makes money is in a way a "negative cycle" in a currency graph, in which currencies are vertices, and exchange rates correspond to weights. One caveat is that all numbers are positive, and they get multiplied rather than added. However, this can be solved by a useful trick: make weights a logarithm of the actual currency exchange value, rather than a value itself. Then logs of numbers less than 1 become negative weights, and adding logs corresponds to multiplying numbers.