MATH 409 LECTURE 10
DIJKSTRA'S ALGORITHM FOR SHORTEST PATHS

REKHA THOMAS

In this lecture we see the first algorithm to compute shortest paths from a given vertex r to all vertices v in a digraph G without negative cost cycles. See [1] and [2] for more details.

The digraph G is assumed to be connected and simple. The algorithms we will see are based on the following principle that we proved last time.

Bellman's principle: Let G be a digraph with all the above assumptions. If $e = vs$ is the final edge of a shorest path $P_{[r,s]}$ from r to s, then $P_{[r,v]}$, which is $P_{[r,s]}$ without $e = vs$ is a shortest path from r to v.

We will look for an algorithm that computes the shortest (r, v)-dipath **for all** $v \in V(G)$. This of course includes the shortest (r, s)-path in G. Note that if we adopt this strategy, then we can store a shortest path from r to v by simply storing the last edge in the path. The shortest path between any two vertices can be reconstructed from this.

Dijkstra's algorithm (Dijkstra 1959)
This algorithm requires that $c_e \geq 0$ for all $e \in E(G)$.
Input: A digraph G with edge costs $c_e \geq 0$ for all $e \in E(G)$. A start vertex r.
Output: The shortest (r, v)-dipaths in G for all $v \in V(G)$ and their costs.

What we in fact get are the following: for each $v \in V(G)$,
— $l(v)$, the cost of a shortest (r, v)-dipath in G
— $p(v)$, the vertex just before v on this shortest (r, v)-dipath.
We will get $l(v) = \infty$ if v is not reachable from r.

(1) Set $l(r) = 0$, $l(v) = \infty$ for all $v \in V(G)\backslash\{r\}$ and $R = \emptyset$. R is the list of vertices processed so far.
(2) Find a $v \in V(G)\backslash R$ with $l(v) = min\{l(w) : w \in V(G)\backslash R\}$.
(3) Set $R = R \cup \{v\}$.

Date: April 19, 2010.

(4) For all $w \in V(G)\backslash R$ such that $vw \in E(G)$ do:
if $l(w) > l(v) + c_{vw}$ then set $l(w) := l(v) + c_{vw}$ and $p(w) := v$.
(5) If $R \neq V(G)$ then go to (2).

Exercise 1. [1, Problem 2.36] Give an example to show that Dijkstra's algorithm can give incorrect results if negative edge costs are allowed.

Recall that all our graphs have n vertices and m edges.

Theorem 2. (1) *Dijkstra's algorithm runs in $O(n^2)$ time.*
(2) *Dijkstra's algorithm works correctly.*

Proof. (1) There are n iterations in the algorithm. The main work in each iteration is done in Step (4) of the algorithm where we update the l-value of all the neighbors of the vertex v being processed in that iteration. Each update takes a constant amount of time and there are at most $n-1$ neighbors for any given vertex. Therefore, the total time taken in $O(n^2)$.

(2) In order to show that the algorithm works correctly, we will argue that the following three properties hold at the end of every iteration.

(a) For all $v \in R$ and $w \in V(G)\backslash R$, $l(v) \leq l(w)$.
(b) For all $v \in R$, $l(v)$ is the length of the shortest (r, v)-path. If $l(v) < \infty$, then there exists an (r, v)-path of length $l(v)$ with final edge $(p(v), v)$ (unless $v = r$ in which case there is no previous node for r) whose vertices are in R.
(c) For all $w \in V(G)\backslash R$, $l(w)$ is the length of a shortest (r, w)-path in $G[R \cup \{w\}]$. If $l(w) < \infty$ then $p(w) \in R$ and $l(w) = l(p(w)) + c_{(p(w),w)}$.

If we prove this then (b) will hold at the end of the algorithm which will prove the correctness of the algorithm.

At the start of the algorithm, $R = \emptyset$ and so (a) and (b) are vacuously true. Let's check that (c) is also true. For $w = r$, $l(w) = 0$ and $G[R \cup \{w\}] = G[\emptyset \cup \{r\}] = G[\{r\}]$. The shortest (r, r)-path in $G[\{r\}]$ does indeed have length zero. For $w \neq r$, $l(w) = \infty$ and $G[R \cup \{w\}] = G[\{w\}]$. There no (r, w)-path in $G[\{w\}]$ which shows that the length of a shortest (r, w)-path in $G[\{w\}]$ is indeed ∞.

Suppose the vertex v is chosen in Step (2) of some iteration of the algorithm and suppose further that properties (a)-(c) hold until the end of the previous iteration. We prove below that Steps (3) and (4) in the current iteration will continue to preserve (a)-(c) showing that these properties will hold again at the end of this iteration.

(a) Pick an $x \in R$ and a $y \in V(G)\backslash R$, $y \neq v$. Suppose we are in Step (2) of the current iteration and have just chosen v to process. Then after Step (2) we have $l(x) \leq l(v)$ since (a) held at the end of the previous iteration and $l(v) \leq l(y)$ since otherwise we would not have chosen v to process. Step (3) does not affect any l-values and so these relations continue to hold. In Step (4), $l(y)$ could change for all $y \in V(G)\backslash R$ such that $vy \in E(G)$. If $l(y)$ changes, it changes to $l(v) + c_{vy} \geq l(v)$ since $c_{vy} \geq 0$. Therefore we still have that $l(v) \leq l(y)$ for all $y \in V(G)\backslash R$, $y \neq v$. Putting everything together, we get that (a) holds at the end of this iteration.

(b) We now check that (b) holds at the end of this iteration. Since this is a statement for vertices in R and (b) held until the end of the previous iteration, we only have to check that (b) holds for the new vertex v that was added to R in Step (3). In Step (3) we add v to R and our job is to check that $l(v)$ is the length of a shortest (r, v)-path in the whole graph. By our assumptions, property (c) held until before Step (3) which implies that $l(v)$ is the length of a shortest (r, v)-path using the vertices in the current R (which includes v). So the only way (b) can fail is if there exists an (r, v)-path P in G using a vertex in $V(G)\backslash R$ that is shorter than $l(v)$. Suppose such a path exists.

Let w be the first vertex in $V(G)\backslash R$ on this path P as you traverse the path from r to v. Since (c) was true before Step (3), $l(w) \leq c(P_{[r,w]})$. Further, since $c_e \geq 0$, $c(P_{[r,w]}) \leq c(P = P_{[r,v]})$ and by assumption, $c(P = P_{[r,v]}) < l(v)$. Putting all this together,

$$l(w) \leq c(P_{[r,w]}) \leq c(P = P_{[r,v]}) < l(v)$$

which implies that $l(w) < l(v)$ contradicting the choice of v in Step (2) as the next vertex to be processed.

(c) Let w be a vertex in $V(G)\backslash R$ at the end of Steps (3) and (4). There are two possibilities: either $l(w)$ got updated in Step (4) or not. If $l(w)$ is not updated, then it was either because, $vw \notin E(G)$ or because $vw \in E(G)$ but $l(w) < l(v) + c_{vw}$.

Case 1. $l(w)$ **is not updated in Step (4) because** $vw \notin E(G)$**:**
Suppose there is a path P in $G[R \cup \{w\}]$ with $c(P) < l(w)$. Then this path must use v since otherwise, it would use only the vertices in $R\backslash\{v\}\cup\{w\}$ and (c) would have been violated in the previous iteration of the algorithm. Also, since $vw \notin E(G)$, there is a node $x \in R$, $x \neq v$ that is the neighbor of w on the path P. These two facts and the fact

that all edge costs are non-negative imply that $c(P) \geq l(v)+c_{xw}$. However, at the start of this iteration (in Step (2)) we had that $l(x) \leq l(v)$ by (a). This implies that $c(P) \geq l(v) + c_{xw} \geq l(x) + c_{xw}$. This last quantity $l(x) + c_{xw} \geq l(w)$ since $l(w)$ was checked for updates when x was processed. Together this implies that $c(P) \geq l(w)$ which contradicts our starting assumption. Therefore, (c) is true in this case.

Case 2. $l(w)$ **is not updated in Step (4) because** $l(w) < l(v) + c_{vw}$**:** In this case, there is an (r, w)-path in $G[R\backslash\{v\} \cup \{w\}]$ with cost $l(w)$ that was a shortest (r, w)-path in this smaller induced subgraph. Adding v to R did not give us a shorter path and so (c) is true for such a w.

Case 3. $l(w)$ **got updated in Step (4):** Then $p(w)$ is set to v and $l(w)$ has been updated to $l(v) + c_{vw}$ and there is an (r, w)-path in $G[R \cup \{w\}]$ of length $l(v) + c_{vw}$ with final edge vw. Suppose there is an (r, w)-path P in $G[R \cup \{w\}]$ which is shorter than $l(w)$. Then this path P must contain v, the only new vertex added to R since otherwise, (c) would have been violated for this w in the previous iteration which contradicts our assumption. (Note that l-values only decrease as the algorithm runs.) Let x be the neighbor of w in this path P. Since $x \in R$, by part (a) we have that $l(x) \leq l(v)$. Also, since $x \in R$ and $xw \in E(G)$, we would have updated $l(w)$ when we processed x which means that $l(w) \leq l(x) + c_{xw}$. Putting all this together, we have

$$l(w) \leq l(x) + c_{xw} \leq l(v) + c_{xw}.$$

However, since P contains v, the length of $P_{[r,v]}$ is at least $l(v)$ which is the length of a shortest (r, v)-path in G. Also, P contains xw and $c_{xw} \geq 0$. Together we get that $c(P) \geq l(v) + c_{xw}$. Including this at the end of the above chain of inequalities we get that $l(w) \leq c(P)$ which contradicts our assumption that $c(P) < l(w)$. □

Exercise 3. [1, Problem 2.18] Show by an example that a spanning directed tree rooted at r can be of minimum cost but not contain least cost dipaths to all the nodes. Also show the converse that it may contain least cost dipaths but not be of minimum cost.

Exercise 4. [1, problem 2.24] There are certain street corners in Bridgetown such that the street on which a car leaves the intersection may depend on the street on which it entered the intersection (for example "no left turn"). How can a digraph, and arc costs, be defined so that the dipaths correspond to legal routes?

Exercise 5. [1, problem 2.21] Run Dijkstra's algorithm on the following digraph. $V = \{r, a, b, d, f, g, h, j, k\}$ and for each $v \in V$ the elements of the list L_v are the pairs (w, c_{vw}) for which $vw \in E$. L_r : $(a, 2)$, $(k, 7)$, $(b, 5)$. L_a : $(d, 8)$, $(f, 4)$. L_b : $(k, 3)$, $(f, 2)$. L_d : $(h, 5)$. L_f : $(g, 3)$, $(j, 7)$. L_g : $(h, 4)$, $(j, 3)$. L_j : $(k, 4)$, $(h, 3)$. L_k : $(d, 2)$, $(h, 9)$, $(g, 6)$, $(f, 1)$. $L_h = \emptyset$.

References

[1] W. Cook, W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics, 1998.
[2] B. Korte and J. Vygen. *Combinatorial Optimization*. Springer, Berlin, 2000.