Algorithmic Combinatorics (Discrete Mathematics II) WS 2018 / 2019
Tibor Szabó

Lightest path, lightest cycle

These are some rough informal notes—to augment your own notes.

Dijkstra's Algorithm A spanning tree Kruskal's Algorthm builds is the lightest possible substructure that is connected (given a non-negative weight function). This might be the optimum for a town to achieve the minimum goal of a connected transportation network as cheap as possible, but also has a couple of disadvantages.

- A tree is a pretty unstable network: the break-down of any link disconnects the whole network. (Bonus HW(?): What is the lightest network that remains connected after the breakdown of any edge?)
- It might take a very long time to travel between some pairs of vertices, because there is a unique path between any pair and that particular path might travel around a bunch of other vertices.

We take a look at the second issue from the point of view of fixed vertex $u \in V(G)$ of some (di)graph/network G with a non-negative weight function $w : E \to \mathbb{R}_{\geq \infty}$ on the edges, and try to find the lightest (=shortest/fastest/cheapest) path from u to another vertex v. Here the weight of a path P is just the sum of the weight of its edges. In fact, given the vertex u, our solution will provide the answer for *any* other vertex $v \in V(G)$. This is Dijkstra's Algorithm, a generalization of BFS.

Algorithm `Dijkstra`

Input: Graph $G = (V, E)$, weight function $w : E \to \mathbb{R}_{\geq 0}$, and starting vertex $u \in V(G)$. Extend w to $\binom{V}{2}$ by setting $w(xy) = \infty$ for each non-edge $xy \notin E$.

Idea: maintain a set $W \subseteq V$ of vertices to which the lightest path is known and add one vertex in each iteration. Maintain a *tentative weight* function $t : V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ storing the weight of the lightest path that was found *so far*. Maintain a function $prev : V \setminus \{u\} \to V$, storing the vertex $prev(z)$ which precedes z on a u, z-path of weight $t(z)$.

Initialization: $W = \{u\}$, $t(z) = w(uz)$, $prev(z) = u$ for every $z \in V \setminus \{u\}$.

Iteration:

- set $v_0 := \arg\min\{t(v) : v \in V \setminus W\}$ (breaking ties arbitrarily)
- update $W := W \cup \{v_0\}$
- for all $v \in V \setminus W$ IF $t(v_0) + w(v_0v) < t(v)$ THEN update $t(v) := t(v_0) + w(v_0v)$ and $prec(v) := v_0$

Termination IF $W = V$ THEN stop and for every $v \in V$ output $t(v)$ (as the weight of the lightest path from u to v) and $v, prev(v), prev(prev(v)), \ldots, u$ (as a lightest uv-path (written in reverse order)).

Theorem 1. *Dijkstra's Algorithm terminates in $n-1$ iterations and outputs what it promises it does.*

Proof. We prove by induction on the number i of iteration steps the following.

Claim 1. *At the beginning of the ith iteration step, we have that*

- (i) $|W| = i$
- (ii) *for every vertex $v \in V$, the vertices $v, prev(v), prev(prev(v)), \ldots, u$ form a u, v-path of weight $t(v)$ (written in reverse order), and all these vertices except v are in W*
- (iii) *for any vertex $z \in V \setminus W$ with $t(z) = \min\{t(v) : v \in V \setminus W\}$ (in particular also for the vertex v_0 that is put into W in the iteration step), the tentative distance $t(z)$ is equal to the lightest possible weight $s(u, z)$ among all u, z-path*
- (iv) *for every $y \in V \setminus W$ and $u \in W$, the tentative distance $t(y) \leq t(u) + w(uy)$*

We prove the Claim soon, let us first derive the theorem. From part (i) it follows that the stopping rule $W = V$ occurs exactly before iteration $|W| = n$, so the number of iteration steps is $n - 1$. Applying part (iii) for the vertex v in the round when it is put into W gives us that $t(z)$ is equal to $s(u, v)$ then. Since vertices that are in W remain there and t does not change on vertices of W, the value is also correct at the end. Finally, part (ii) together with part (iii) proves that the path in question is a lightest possible one. □

The technical part (iv) is not needed anywhere to derive the final proposition, but it is an essential part for the induction step to go through. This phenomenon, the "loading of the induction hypothesis" is a general and useful technique of inductive proofs. It becomes necessary, when the inductive statements we are interested in proving (here the parts $(i) - (iii)$) do not seem strong enough to prove the induction step for themselves (for example, here we are unable to derive the statements $(i) - (iii)$ for iteration step $i + 1$ just from the knowledge that the statements are true for the iteration step i). The art of loading the induction hypothesis constitutes of identifying a "stronger" (true) statement (i.e. adding part (iv)) in a way that it is now strong enough to help the proof of the induction step for the "interesting" parts (i.e. the parts $(i) - (iii)$), but is not *too* strong so that we are now unable to prove the induction step for itself (i.e. we should be able to prove part (iv) for iteration step $i + 1$ from the knowledge that parts $(i) - (iv)$ are true for iteration step i). We will see other nice examples of this phenomenon later in the course as well.

The proof of the Claim relies on the following simple fact.

Key Observation: *If P is a x, y-path of weight $s(x, y)$, then for any vertex $z \in V(P)$, the x, z- and z, y-segments of P have weight $s(x, z)$ and $s(z, y)$, respectively. In particular, $w(uv) = s(u, v)$ for any edge $uv \in E(G)$.*

Proof. If there were a lighter path Q from x to z than $P[x,z]$, then the concatenation of Q and $P[z,y]$ would form an x,y-walk R that is lighter than P. Any x,y-walk, hence R as well, contains an x,y-path R' with $E(R') \subseteq E(R)$. Since the weight function is non-negative, $w(R') \leq w(R) < w(P)$, a contradiction to the minimality of P. □

Proof of Claim. Before the first iteration all conditions hold due to the initial choices. The set W has $|\{u\}| = 1$ element, for every vertex $v \in V \setminus \{u\}$ the vertices $u = prev(v), v$ form a u,v-path of weight $t(v) = w(uv)$. For part (iii) note that if $t(z) = w(uz)$ is minimum among vertices z of $V \setminus \{u\}$, then every u,z-path has at least this weight (since it already picks up weight at least $w(uz)$ with its starting edge and further edges cannot decrease the weight since the weight function is non-negative) and consequently $s(u,z) = w(uz) = t(z)$. Part (iv) holds since $t(y) = w(uy) = t(u) + w(uy)$ for every $y \in V \setminus \{u\}$.

For the induction step assume that $(i) - (iv)$ hold before iteration $i \geq 1$. Part (i) follows immediately, since in iteration i exactly one vertex is moved from $V \setminus W$ to W and no vertex is moved out.

For (iii) suppose that $z \in V \setminus W$ is a vertex with $t(z) = \min\{t(v) : v \in V \setminus W\}$, yet $t(z) > s(u,z)$. Take a lightest u,z-path P in K_n, so P has weight $s(u,z)$. Let v_f be the first vertex on P that is not in W. (Note that there is at least one such a vertex, as $z \notin W$.) And let v_p be the vertex immediately preceding v_f on P. By our Key Observation and the non-negativity of the weight function, we have $t(z) > s(u,z) = s(u,v_p) + w(v_pv_f) + s(v_f,z) \geq s(u,v_p) + w(v_pv_f)$ Using part (iii) of the induction hypothesis for the iteration step when the vertex v_p was put into W, we have $s(u,v_p) = t(v_p)$. Putting this in the above, and then using part (iv) for $v_f \in V \setminus W$ and $v_p \in W$, we have $t(z) > t(v_p) + w(v_pv_f) \geq t(v_f)$. This is a contradiction to the minimality of $t(z)$.

Part (ii) follows immediately from the updating rule and the induction hypothesis. If $t(v)$ is not updated then the statement follows by induction and the fact that vertices are never removed from W. Suppose now that $t(v)$ is updated to $t(v_0) + w(v_0v)$ and $prev(v)$ is updated to v_0. By the induction hypothesis vertices $v_0, prev(v_0), prev(prev(v_0)), \ldots, u$ formed a u,v_0-path of weight $t(v_0)$ before iteration i, and all except v_0 was in W. None of these, and $t(v_0)$ and $prev(v_0)$ are updated in the ith iteration, so part (ii) holds for v after iteration i.

To prove (iv) before iteration $(i+1)$, let $y \in V \setminus (W \cup \{v_0\})$ and $u \in W \cup \{v_0\}$. If $u = v_0$ then the statement holds because of the updating during iteration step i. If $u \in W$, then the statement holds by induction hypothesis. □

The running time of an iteration step is $O(n)$, so overall the algorithm runs in $O(n^2)$ time.

Traveling Salesman Problem Given a graph with (non-negative) edge weights we gave algorithms to find a lightest spanning tree (Kruskal) and path between two given vertices (Dijkstra). Now we will be concerned with finding the lightest Hamilton cycle.

Example Euro 2020 organized by 13 countries: optimization problem for (poor (English)) football fans.

In the Traveling Salesman Problem (TSP) we are given a weight function $w : \binom{[n]}{2} \to \mathbb{R}_{\geq 0}$, and we aim to find the lightest Hamilton cycle.

The brute force algorithm would consider all the $(n-1)!$ Hamilton cycles, calculate its weight and put out the smallest one. This would take at least $\Theta(n)(n-1)!) = \Theta\left(\frac{n^n}{e^n}\sqrt{n}\right)$ step, a humongous number even for relatively small values of n. (For a graph on just 70 vertices this would take roughly 10^{100} steps (way more than the number of atoms in the observable universe).

One reason the brute force algorithm takes so long is that after checking one Hamilton cycle H, it "forgets" a lot of information it calculated about the length of paths that are segments of H and in principle could be used when the weight of another Hamilton cycle, also containing the segment, is calculated. Based on this observation one can try to create a recursive algorithm structure that, in order to save time, stores and reuses these already calculated segment length. This idea is the basis of the *dynamic programming* method. The Held-Karp Algorithm does exactly this and its running time is "only" $O(n^2 2^n)$, much less than the running time of the brute force algorithm.

The disdvantage of the dynamic programming algorithm is that it must store a lot of information, the Held-Karp Algorithm for exmple needs to use at least $\Omega(2^n)$, so exponential amount of space. The much slower brute force algorithm on the other hand uses only quadratic amount of space (since after calculating the weight of each Hamilton cycle, the calculated data is erased).

Despite a significant effort by many researchers since the 1960's, and the development of substantially different, tricky approaches that can solve concrete problem instances up to tens of thousands of vertices (which is waaay beyond the applicability of the Held-Karp Algorithm), the theoretical time bound of $2^n n^2$ was not improved significantly ever since. That means that we do not even know for example whether there is an algorithm which works in 1.999^n steps. And you can imagine, considering the fundamental nature and applicability of the problem, that this is NOT due to humanity's lack of trying...