Chapter 36

Minimum Directed Spanning Tree

By Sariel Har-Peled, August 31, 2023[1] Version: 0.3

> Blessed with so many resources within myself, the world was not necessary to me. I could do very well without it.
>
> Emma, Jane Austen

. .

36.1. Preliminaries

Given a directed graph G with a root r, a ***directed spanning tree*** (for r) is a directed tree $\mathcal{T} \subseteq \mathsf{G}$ rooted at r, such that there is a path from r to all the vertices of G. (Alternatively, one can start with a spanning graph with this property, and consider a minimal subgraph with this property, which is clearly a tree.) It is thus a directed rooted tree routed at r. This is usually called in the literature an ***arborescence***, but since I neither know how to pronounce this word, or spell it (without cut and paste), I would use the more natural term ***DST*** (i.e., directed spanning tree).

Computing a DST for r can be readily done using BFS/DFS. The more interesting question is naturally compute the minimum weight such tree, when there are non-negative weights on the edges – specifically, for an edge $e \in \mathsf{E}$, lets its weight is $\omega(e) \geq 0$. This tree is the ***minimum directed spanning tree*** of G for r (or the ***MDST*** of G and r).

This is the directed version of MST, but computing it is somewhat more challenging. The MDST is the cheapest tree containing a path from r to all the other vertices in G. It is thus the cheapest broadcast tree in G.

Lemma 36.1.1. *Let $\mathcal{T}$ be a DST of G and r. We have the following properties:*

(i) The node r has no incoming edge in $\mathcal{T}$.

(ii) Every vertex $v \in \mathsf{V}(\mathsf{G}) - r$, has exactly one incoming edge.

(iii) A graph $\mathsf{H} \subseteq \mathsf{G}$ with no (directed) cycles, and with the above two properties is a DST.

Proof: (i) If there was one, then we would delete it.

(ii) There must be at least one incoming edge, as there is a path in $\mathcal{T}$ from r to v. If there are more than one incoming edge, then we can delete all of them except one. Clearly, the resulting graph is still a DST of G for r.

(iii) Consider a vertex $v \in \mathsf{V}(\mathsf{G}) - r$. Set $v_1 = v$, and in the ith stage, consider the (unique) incoming edge (v_{i+1}, v_i). If $v_{i+1} = r$, the we proved that there is a path from r to v in G. Otherwise, continue to the next stage. By the finiteness of G, either this process reaches r, or we encounter a cycle, where $v_i = v_j$, for $j < i$. But the later one is impossible. We conclude that H contains a path from r to v, for all $v \in \mathsf{V}(\mathsf{G})$.

The graph H can not have a cycle (if we ignore the directions of the edges), as the edges of such a cycle must be all directed in a consistent way around a cycle, as otherwise there would be two edges incoming into a single vertex. Indeed, the direction of a single edge in the cycle, forces the direction of all the edges in the cycle, but this then would form a directed cycle in H, contradicting assumption. As such, H is indeed a tree. ■

36.2. Towards an algorithm

Since every vertex (except r) in a DST has a unique edge coming into it, it is natural to move the pricing from the edges to the vertices. In particular, every vertex will pay for the edge incoming into it in the MDST. Clearly, this would cover the price of the MDST. A natural first step, similar in spirit to Borůvka's algorithm, is for every vertex $v \in \mathsf{V}(\mathsf{G}) - r$ to choose the cheapest incoming edge. This results in a set of edges F. We refer to the graph (V, F) as the ***frame*** of G.

Lemma 36.2.1. *If the frame (V, F) is a DST for r, then it is the MDST.*

Proof: Every vertex of $\mathsf{V} - r$ has to pay at least the price of the cheapest edge coming into it in the MDST. The price of the (V, F) is exactly this price. ∎

If the above fails, then (V, F) must contain a cycle C (which can not contain r). The idea is to collapse this cycle to a vertex, compute the MDST in the collapsed graph G/C, and then take the solution MDST for this graph and expand it back to MDST to the original graph – by adding back all the edges of the cycle, except one. The challenge of course is to decide which edges of the cycle should we add back.

To this end, for a vertex v, let y_v be (intuitively) the price the vertex v is willing to pay directly for its incoming edge. We set

$$\forall v \in \mathsf{V} \qquad \mu_v = \min_{e=(u,v)\in\mathsf{E}} \omega(e).$$

This is the maximum a vertex can pay for before the prices of the edges incoming into it become negative (bad). We select values of y_v to be anywhere between 0 and μ_v. But then one need to adjust the prices of the edges entering each vertex. Formally, let

$$\forall (u,v) \in \mathsf{E} \qquad \omega'(u,v) = \omega(x,y) - y_v.$$

Interestingly, if $y_v = \mu_v$, then each vertex (except r) must have at least one incoming edge of price zero – conceptually this is a "free" edge that we would like to use in the solution. The key observation is that indeed we can use the new pricing of the edges.

Lemma 36.2.2. *Let $\mathcal{T}$ be the optimal MDST for G under ω, and let $\mathcal{T}'$ be the optimal MDST for G under the pricing of ω'. Then, we have that*

$$\omega(\mathcal{T}) = \omega'(\mathcal{T}') + \Delta, \qquad \textit{where} \quad \Delta = \sum_{v\in\mathsf{V}-r} y_v.$$

Proof: Let $\mathcal{T}''$ be any DST for r of G, and observe that for any $v \in \mathsf{V} - r$, there is a single edge of $\mathcal{T}''$ incoming into it. As such, we have

$$\omega(\mathcal{T}'') = \sum_{(u,v)\in\mathcal{T}''} \omega(u,v) = \sum_{(u,v)\in\mathcal{T}''} (\omega'(u,v) + y_v) = \sum_{(u,v)\in\mathcal{T}''} \omega'(u,v) + \sum_{v\in\mathsf{V}-r} y_v = \omega'(\mathcal{T}'') + \Delta.$$

As such, the DST minimizing the price over ω or ω' is the same . ∎

This idea of moving prices around is especially useful if one can create a cycle where all the edges have price zero.

Lemma 36.2.3. *Let C be a cycle such that all its edges have price zero under ω', and it does not contain r, where $\omega'(e) \geq 0$, for all $e \in \mathsf{E}$. Then, there a MDST $\mathcal{T}$ of G for r, under the prices of ω', that contains all the edges of C except for one. Namely, there is a MDST of G with a single incoming edge into the vertices of C.*

Proof: Let $\mathcal{T}$ be a MDST of G, and let v_1 be the closest vertex in C to the root of $\mathcal{T}$ (in $\mathcal{T}$). Let $\mathsf{C} = v_1 \to v_2 \to \cdots \to v_k \to v_1$. For $i = 1, \ldots, k-1$, if (v_i, v_{i+1}) is not in $\mathcal{T}$, then we delete the incoming edge into v_{i+1} in $\mathcal{T}$ (it must have cost at least zero), and replace it by (v_i, v_{i+1}). Clearly, the resulting $\mathcal{T}'$ is still a DST, and it is not more expensive than $\mathcal{T}$ was. ∎

36.3. An algorithm

The above lemma suggests to slowly move the prices of the edges of the MDST to the direct prices payed by the vertices. There are several ways to do this, but here is maybe the cleanest. We compute the frame (V, F) of G. If it does not contain a cycle, then we are done. Otherwise, let C be a cycle of G. We set $y_v = \mu_v$, for all $v \in \mathsf{V}(\mathsf{C})$ (i.e., $y_v = 0$ for all other vertices). Under the new pricing ω', the prices of all the edges of C are zero (there might be edges incoming into C with zero price, or the edges between vertices of C with zero price – this is fine).

By Lemma 36.2.3, there is MDST such that cycle has a single incoming edge. As such, the algorithm readjusts the prices of the edges around C, setting $y_v = \mu_v$, for all $v \in \mathsf{V}(\mathsf{C})$, and computes the new adjusted weight function ω'. The algorithm then collapses the cycle C to a single vertex! Let $\mathsf{H} = \mathsf{G}/\mathsf{C}$ be the resulting graph. The algorithm computes recursively the MDST $\mathcal{T}'$ for H using the weights specified by ω'. The algorithm then expands $\mathcal{T}'$ into a MDST of G, by reinserting all the edges of the cycle into it, except for the one that already has an incoming edge.

How to collapse the cycle. Assume $\mathsf{C} = v_1 \to \cdots \to v_k \to v_k$. We create a new meta vertex u, and we replace all edges (x, v_i) by an edge (x, u) of the same price. If there are several parallel edges, we keep only the cheapest one. We update the weights of the edges accordingly.

Correctness. If the frame is a DST, then we are done. Otherwise, there is a cycle, and there is a MDST that has a single incoming edge into C. Under ω', the cycle has price zero, so we can simply collapse it. Clearly, the corresponding collapse MDST is a MDST of H. Thus, computing the MDST of H and "unrolling" it, results in a MDST of G.

Running time. The way we described it, the running time is pretty bad. Computing the frame, finding the cycle, collapsing the cycle, etc, can all be done in $O(n + m)$ time, and it reduces the number of vertices in the graph by at least one. As such, the resulting running time is $O(n(n + m))$.

Theorem 36.3.1. *Given a graph G with n vertices and m edges, a root node r, and with positive weights on the edges, one can compute the minimum directed spanning tree (i.e., MDST or min cost arborescence) of G for r in $O(n(n + m))$ time.*

36.4. Bibliographical notes

The fastest algorithm known for this problem is due to Gabow *et al.* [GGST86]. The idea is to do the adjacent in a more localized fashion, essentially emulating Prim's algorithm. As in Prim's algorithm, one start with a graph where every vertex is a singleton. Then, one picks a vertex, and repeatedly look for the cheapest incoming edge into this vertex. If such an edge is found, the edge is added to the constructed graph, and the algorithm moves to this next vertex. This builds a path. If the path arrives to the root, then we collapse all the vertices along the path to the root, and we continue. The problematic case is when the incoming edge comes from a vertex earlier in the path. This corresponds to a cycle. We collapse the cycle into a single vertex and continue. The main idea being that we maintain implicitly the set of all incoming edges into the current active vertex (which might now be a connected component). To maintain this efficiently under the above changes, one needs to use heap on the edges, and dynamically maintain the price adjustment as we merge connected components on the fly. The details are somewhat involved – the interested reader should check out [GGST86].

Bibliography

[GGST86] Harold N. Gabow, Zvi Galil, Thomas H. Spencer, and Robert Endre Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Comb.*, 6(2):109–122, 1986.