& All Pairs

☐ SS-ST: Speeding up Dijkstra!? Wagner & Willhalm paper
☐ Preview (only) of All-Pairs Shortest Paths Ch.25

Speeding up Dijkstra!?

- Binary heaps ⇒ Fibonacci Heaps helps
we perform |V| Extract-Min's
and up to |E| Decrease-Key's, but we
do not care how much each takes in
the worst case; we only care about the
total (for a worst-case bound on Dijkstra).
Fibonacci Fibonacci heaps perform |E|
Decrease-Key operations in O(|E|) time,
so total for Dijkstra is O(V |V| g V + E).
- It would be hard to do better, because we need to consider all the edges, and we extract vertices in sorted order (distance from s), so we essentially sort.

![Graph Diagram]

- If we input a graph like this to Dijkstra, it will process the edges/vertices in sorted order.

- Still, people invested effort in speed-up techniques that do not change the worst-case asymptotic running time, but the constants and/or the average or "common" case.

- How much more would you pay for a computer that was "only" x2 faster? Probably something.

- Asymptotic worst-case improvements are some of the most valuable, but there is value in weaker results.
Run two copies of Dijkstra simultaneously, alternating steps, one from s on G, and one from the target t on G_{rev}=(V, \{(u,v): (u,v) \in E\}). Stop when some vertex is extracted from both heaps.
\[ d[w] = \begin{cases} 3 & \text{if } \text{Rev}[w] = 3 \\ 6 & \text{if } \text{Rev}[w] = 6 \\ 10 & \text{otherwise} \end{cases} \]

\[ d[v] = 3 \]

\[ d[x] = 5 \]

\[ d[y] = 5 \]

\[ d[t] = 10 \]

processed in both \( G \) & \( \text{Rev} \), stop.

\[ d[E] \text{ is not the shortest path!} \]

But \[ \min \{ d[w] + d[\text{Rev}[w]] \} \text{ is.} \]

**Proof:**

\[ \delta(s,w) > \delta(s,x) \]

\[ \delta(w,t) > \delta(x,t) \]

\[ \Rightarrow \text{swuxt longer than swxust} \]

But we do have the correct weights of \( s \rightarrow u \rightarrow v \) and \( u \rightarrow v \rightarrow t \)

\( s \rightarrow u \) and \( u \rightarrow v \)

Simple idea but a tricky detail; beware.
Single-Source Single Target using Potentials

Suppose the vertices represent points in the plane (e.g., on a map) and \( w(u,v) \) is the Euclidean distance from \( u \) to \( v \). (Edges are straight lines).

E.g., you start walking from 5th Av & 52nd St in NY to Albany, NY north. Should the next vertex be 51st & 5th or 53rd & 5th (it's a tie on the map).

We'll show you can do the logical thing and not increase the asymptotic running time (maybe reduce it a bit).
Let $p(w) = \| w - t \|$ (Euclidean distance from $w$ to $t$).

Define $\hat{w}(u,v) = w(u,v) - p(u) + p(v)$

\[
\begin{align*}
&\hat{w} = 0 \\
&u \quad w = 2 \\
&v \quad w = 2
\end{align*}
\]

$\hat{w} \geq 0$, can run Dijkstra

$\hat{w}(\text{path}) = \hat{w}(\langle u_1, v_2, \ldots, v_{k-1}, v_k \rangle)$

\[
= \sum_{i=2}^{k} \hat{w}(v_{i-1}, v_i)
\]

\[
= \sum_{i=2}^{k} w(v_{i-1}, v_i) - p(v_{i-1}) + p(v_i)
\]

\[
= p(s) + p(t) + \sum_{i=2}^{k} w(v_i, v_{i+1})
\]

\[
= w(\text{path}) - p(s)
\]

relative rank of paths is preserved
All Pairs Shortest Paths (Preview)

Clearly more expensive (at least not cheaper) but there is so much to compute that algorithms become simpler.

Floyd-Warshall: $O(v^3)$

- Set up an $n \times n$ matrix $D$
- Initialize $D_{ij} = w_{ij}$ length of SP that do not go through any intermediate vertex
- Shorten paths by going through vertex 1, if possible
- Repeat for vertices 2 through $n$.

Johnson: $O(V^2 \log V + EV)$

Run Dijkstra from every vertex after finding a potential $p$ that makes edges non-negative (using Bellman-Ford)

Using building blocks