Lecture 15: All Pairs Shortest Paths: Johnson's Algorithm

Today: All-Pairs Shortest Path (APSP) Problem

- Setup: weighted graph $G = (V, E, w)$
- **APSP Problem Statement:** Compute $\delta(u, v)$ for all $u, v \in V$, as well as a shortest path tree from each node.
  - The output has $O(V^2)$ size.
  - If there's a negative cycle, abort.
- Useful when need to understand a network as a whole. E.g., transportation, circuit layout, supply-chain management, etc.
- Easy solutions: SSSP from each node
  - If unweighted, run BFS from each node: $V \times O(V + E) = O(V^2 + VE)$
  - If nonnegative weights, run Dijkstra from each node: $V \times O(V \log V + E) = O(V^2 \log V + VE)$
  - With negative weights allowed, run Bellman-Ford from each node: $V \times O(VE) = O(V^2 E)$
- Let's do better!

Adjusting Edge Weights

- Let's focus on graphs with negative weights allowed
- Idea: can we modify the weights to be nonnegative? Then we could use Dijkstra instead of Bellman Ford!
- Some attempts to modify edge weights while preserving shortest paths:
  - Add $1{,}000{,}000$ to each edge weight. Are shortest paths still shortest paths? (No, we're biasing paths with fewer edges.)
  - Different idea: pick a node u, add 10 to its incoming edges, and subtract 10 from its outgoing edges. Are shortest paths still shortest paths? (Yeah!)
- **General Strategy**: take any function $h : V \to \mathbb{R}$ defined on **vertices**, and set new edge weights $w'(u, v) = w(u, v) + h(v) - h(u)$. Call the new graph $G' = (V, E, w')$.

  - Let's call h a **reweighting function**. (Sometimes called a "heuristic" or "potential".)
  - Claim: if p is a path from u to v, then $w'(p) = w(p) + h(v) - h(u)$. In other words, *all* paths from u to v are adjusted by the same amount. Proof: the h terms telescope.
  - Claim: if p is a path from u to v, then p is shortest in G iff it is shortest in G'. Proof: by previous claim, all u-v paths are adjusted by the same amount, $h(v) - h(u)$.
  - Result: A shortest path tree from s in G' is automatically a shortest path tree from s in G.

- So if we could solve Single-Source, All-Pair, or Single-Pair shortest path problem on G', that automatically solves the same problem for G.

- Can we find a reweighting function h that makes these new weights ≥ 0?

Making All Weights Nonnegative

- Clever choice of reweighting function: Add a new node X with weight-0 directed edges (X, v) for each $v \in V$, and choose $h(v) = -\delta(X, v)$.

- Observation: $\delta(X, v)$ is always ≤ 0 (why?), so $h(v) \geq 0$.

- Pro: the new weights $w'(u, v) = \delta(X, u) + w(u, v) - \delta(X, v)$ are ≥ 0 by Triangle Inequality! Hooray!

- Con: Computing all $h(v)$ requires solving SSSP with negative edge weights! E.g., Bellman-Ford

- So this doesn't help us improve on Bellman-Ford :(But it does help with APSP!

Johnson's Algorithm

- **Johnson's Algorithm** solves All-Pairs Shortest Paths on a graph with signed weights, by **reducing** to a graph with nonnegative weights

- Idea: Reweight, then V rounds of Dijkstra

- Algorithm details:

  - Add new vertex X and weight-0 edges (X, v) for all $v \in V$ as above, and run Bellman-Ford from X to compute $h(v) = -\delta(X, v)$. Use these to form a new graph G', which will have **nonnegative weights**. (If Bellman-Ford detects a negative weight cycle, abort.)
  - Run Dijkstra V times on G', once from each node, to find a shortest path tree and shortest path weights $\delta'(u, v)$ in G'

  - The correct δ values in G are $\delta(u, v) = \delta'(u, v) + h(u) - h(v)$, and the shortest path trees in G' are also shortest path trees in G

- Runtime: $O(VE)$ for Bellman-Ford, $O(V^2 \log V + VE)$ for V runs of Dijkstra, and $O(V^2)$ to compute δ from δ'. Total is $O(V^2 \log V + VE)$. As good as for positive-weight graphs!

- Correctness:

  - Already proved that G' has nonnegative weights, that shortest path trees on G' work for G, and that δ and δ' are related as claimed.
  - Already proved that Bellman-Ford and Dijkstra behave as required.
  - Done!

- This is a **Reduction**: we're reducing the "Signed APSP" problem to the "Nonnegative APSP" problem, i.e., using the latter as an intermediate step to solving the former. Instead of reinventing entirely new algorithms, we're using familiar ones we already trust. Hence the short correctness proof.

Review of Shortest Path Problems and Algorithms

- We've talked about many shortest path algorithms. How to know which one to use? **Use the most specific one you can** for the given problem, since those are faster. Examples:

  - Need a *single* distance $\delta(s, t)$? That's the Single-Pair shortest paths problem, but we don't have algs that are provably better than SSSP for this, so just use an SSSP alg.
  - Somehow find yourself with a DAG? (Check carefully; it's not always obvious at first glance.) Use DAG relaxation instead of something more powerful.
  - Unweighted graph? Don't need anything fancy; BFS is simplest and fastest.
  - Are you *sure* your graph needs negative edge weights? Only then should you reach for Bellman-Ford or Johnson.
  - Before you reach for an APSP algorithm, are you sure you need distances **from** lots of different nodes *and* **to** lots of different nodes? If you can get away with SSSP instead, use that first.

- In other words, go through the tables below row by row and use the first one that applies.

- Single-Source Shortest Path Algorithms

Graph	Weights	SSSP Algorithm	Runtime (big-O)	Reminder
DAG	Any Real	DAG relaxation	$V + E$	Relax in order
General	Unweighted	BFS	$V + E$	Explore level by level
General	Nonnegative	Dijkstra	$V \log V + E$	Relax in Priority Queue order
General	Any Real	Bellman-Ford	VE	Relax in $V - 1$ rounds

- All-Pairs Shortest Path Algorithms

Graph	Weights	APSP Algorithm	Runtime (big-O)
DAG	Any Real	$V \times$ DAG Relaxation	$V \cdot (V + E) = V^2 + VE$
General	Unweighted	$V \times$ BFS	$V \cdot (V + E) = V^2 + VE$
General	Nonnegative	$V \times$ Dijkstra	$V \cdot (V \log V + E) = V^2 \log V + VE$
General	Any Real	Johnson	$VE + V \cdot (V \log V + E) = V^2 \log V + VE$

- Single-Pair Shortest Path Algorithms
  - No guaranteed asymptotic improvements over Single-Source algorithms
  - Heuristics tend to help in practice:
    - Bidirectional Dijkstra: search from both ends, stop when they meet
    - Many others!

Setting	SSSP Algorithm	Running Time
DAG, any weights	**DAG Relaxation**	$O(V + E)$
Unweighted	**BFS**	$O(V + E)$
Weights ≥ 0	**Dijkstra**	$O(V \log V + E)$
General	**Bellman-Ford**	$O(VE)$

Setting	APSP Algorithm	Running Time
DAG, any weights	$\boldsymbol{V} \times$ **DAG Relaxation**	$O(V \cdot (V + E)) = O(V^2 + VE)$
Unweighted	$\boldsymbol{V} \times$ **BFS**	$O(V \cdot (V + E)) = O(V^2 + VE)$
Weights ≥ 0	$\boldsymbol{V} \times$ **Dijkstra**	$O(V \cdot (V \log V + E)) = O(V^2 \log V + VE)$
General	$\boldsymbol{V} \times$ **Bellman-Ford**	$O(V \cdot (VE)) = O(V^2 E)$

Setting	APSP Algorithm	Running Time
DAG, any weights	$\boldsymbol{V} \times$ **DAG Relaxation**	$O(V \cdot (V + E)) = O(V^2 + VE)$
Unweighted	$\boldsymbol{V} \times$ **BFS**	$O(V \cdot (V + E)) = O(V^2 + VE)$
Weights ≥ 0	$\boldsymbol{V} \times$ **Dijkstra**	$O(V \cdot (V \log V + E)) = O(V^2 \log V + VE)$
~~General~~	~~$\boldsymbol{V} \times$ **Bellman-Ford**~~	~~$O(V \cdot (VE)) = O(V^2 E)$~~
General	**Johnson**	$O(V \cdot (V \log V + E)) = O(V^2 \log V + VE)$

https://qiao.github.io/PathFinding.js/visual

Single-Source and All-Pairs Shortest Paths Algorithms

Setting	SSSP Algorithm	Running Time
DAG, any weights	**DAG Relaxation**	$O(V+E)$
Unweighted	**BFS**	$O(V+E)$
Weights ≥ 0	**Dijkstra**	$O(V\log V+E)$
General	**Bellman-Ford**	$O(VE)$

Setting	APSP Algorithm	Running Time
DAG, any weights	$\boldsymbol{V} \times$ **DAG Relaxation**	$O(V\cdot(V+E))=O(V^2+VE)$
Unweighted	$\boldsymbol{V} \times$ **BFS**	$O(V\cdot(V+E))=O(V^2+VE)$
Weights ≥ 0	$\boldsymbol{V} \times$ **Dijkstra**	$O(V\cdot(V\log V+E))=O(V^2\log V+VE)$
General	**Johnson**	$O(V\cdot(V\log V+E))=O(V^2\log V+VE)$