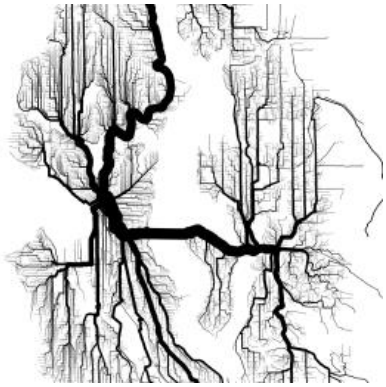
Ma/CS 6a

Class 27: Shortest Paths



By Adam Sheffer

Naïve Path Planning

- **Problem.**
  - We are given a map with cities and *non-crossing* roads between pairs of cities.
  - Describe an algorithm for finding a path that between city A an city B that minimizes the number of roads traveled.

Solution

- Turn the problem into a graph:
  - Every city is a vertex.
  - Every road is an edge.
  - Find a path from vertex A to vertex B that consists of a minimum number of edges.
  - Immediate by running BFS from A.

Slightly Less Naïve Path Planning

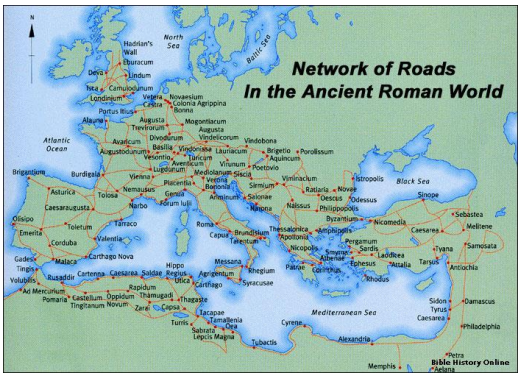
- **Problem.** Same as previous problem, except that roads are allowed to cross (contain intersections).
  - Minimize the number of *road segments* that were traveled.

Solution

- Turn the problem into a graph:
  - Every city *and every intersection* is a vertex.
  - Every *road segment* is an edge.
  - Find a path from vertex A to vertex B that consists of a minimum number of edges.
  - Immediate by running BFS from A.

Path Planning

- **Problem.** Given a map with roads, find a path from city A to city B, that *minimizes the distance* travelled.

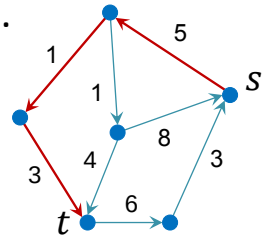


Formulating the Problem

- Turn the problem into a graph:
  - Every city and every intersection is a vertex.
  - Every road segment is an edge.
  - A *weight function* $w: E \to \mathbb{R}$ that gives every edge its distance.
  - Find a path from vertex A to vertex B that *minimizes the sum of its edge weights*.

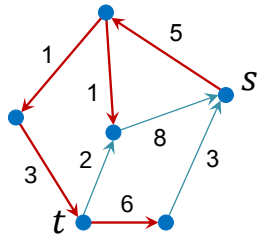
Shortest Paths Problem

- Consider a directed graph $G = (V, E)$ a weight function $w: E \to \mathbb{R}$.
- The *weight* of a path P is the sum of the weights of its edges.
- **Problem.** Given a pair of vertices $s, t \in V$, find a shortest path between s and t (i.e., a path of minimum weight).



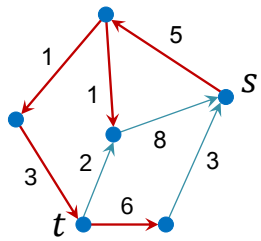
The Plan

- We consider the (common) case where the weights are non-negative.
- We describe a ***greedy*** algorithm for finding the shortest paths from a vertex $s \in V$ to ***every*** other vertex of V.



Subpaths of a Shortest Path

- **Claim.** Consider a directed graph $G = (V, E)$, a weight function $w: E \to \mathbb{R}$, and a shortest path P between the vertices $s, t \in V$.
  - For any vertex $v \in P$, the segment of P between s and v is a shortest path.

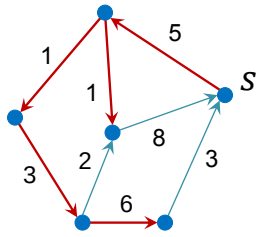


Proof of Claim

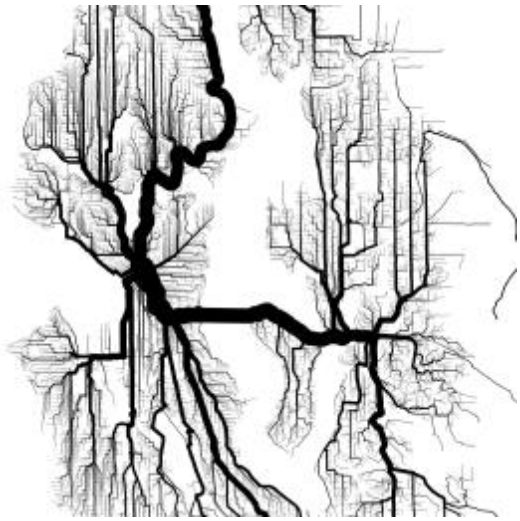
- Assume, for contradiction, that there exists $v \in P$ such that the segment of P from s to v is not a shortest path.
  - P_1 – the segment of P from s to v.
  - P_2 – the segment of P from v to t.
  - ℓ_1, ℓ_2 – the lengths of P_1, P_2, respectively.
  - By assumption, there exists a path P' from s to v of length $< \ell_1$.
  - Combining P' and P_2 yields a path from s to t of length $< \ell_1 + \ell_2$. *Contradiction to P being a shortest path!*

Shortest Paths Tree

- **Corollary.** There exists a tree that contains shortest paths from s to any other vertex that is accessible from it.
- We refer to such a tree as a *shortest paths tree* from s.



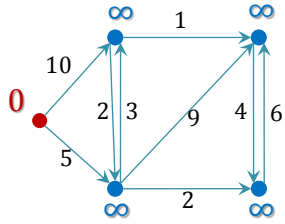
Shortest Paths Tree of Seattle

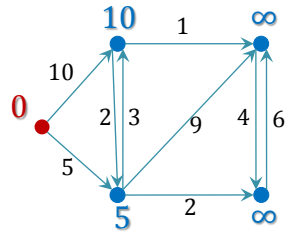


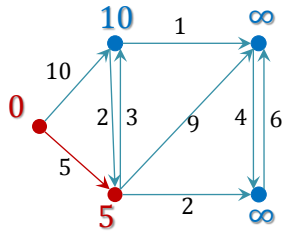
Dijkstra's Greedy Algorithm

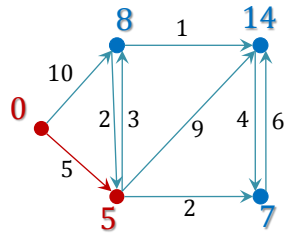
- $d[v]$ – the weight of the shortest path from s to v that we found so far.
- We grow the shortest paths tree in steps.
  - We start by setting $d[s] = 0$.
  - At each step, out of the vertices that are not in the tree, we add the one that minimizes $d[v]$ (in the first step, we add s).
  - After adding a vertex v to the tree, for every $(v, u) \in E$, we check whether
    $$d[u] > d[v] + w(v, u).$$
    If this holds, we set $d[u] = d[v] + w(u, v)$.

Dijkstra Example

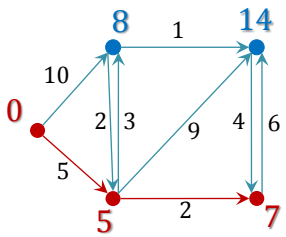


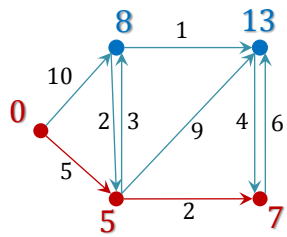


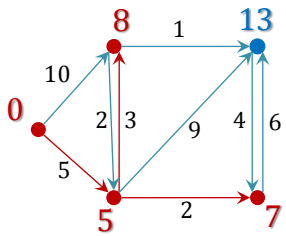


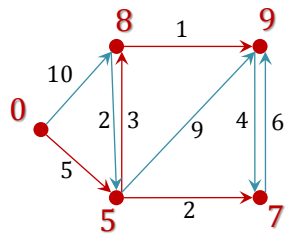


Dijkstra Example (cont.)









Dijkstra Correctness

- The algorithm clearly returns a graph of paths from s to every vertex that is accessible from s.
- **Claim.** The paths are determined by Dijkstra are ordered by increasing weight.
  - At each step, the algorithm chooses the shortest path that it knows about.
  - Every new path that is discovered is at least as heavy as the ones that were previously chosen, since every new path is chosen path plus an additional edge.

Correctness (Partial/Intuitive)

- One can prove, by induction on the number of elements in the tree, that the vertices $v \in V$ are inserted into the tree in order and with correct $d[v]$.
  - The value $d[s] = 0$ is correct by definition.
  - Induction step. Consider a vertex v that was just added to the tree, let P be a shortest path to v, and let u be the vertex before v in P.
  - That is, $d[v] = d[u] + w(u, v)$. By the hypothesis, u is in the tree and the correct $d[v]$ was found when we inserted u to the tree (not accurate).

Travelling Salesman Problem

- **Problem.** Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

December 21, 2011, 9:00 PM

The Problem of the Traveling Politician

By WILLIAM COOK

Politicians have always been heavy travelers, but recent news out of Iowa takes the notion of hitting the campaign trail to new extremes.

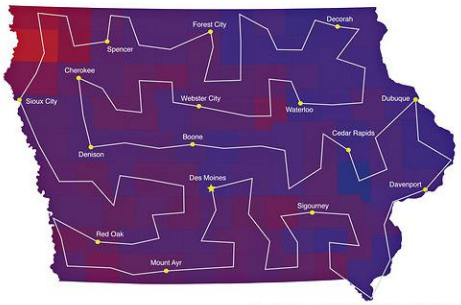
Rick Santorum has toured all 99 counties and he is back criscrossing the state.

Thursday is day seven of Michele Bachmann's own 99-county tour. She plans to cover the state in 10 days, with time off for Christmas.

Rick Perry and Newt Gingrich each announced 44-city bus tours leading up to the January caucuses.

Not to be left out, Mitt Romney announced on Tuesday that he would begin his three-day tour on Dec. 28.

With all this movement, we thought we might lend a hand and offer a suggestion for saving time and gasoline.



W.J. Cook, A.L. Kornhauser, and R.J. Vanderbei

The Naïve Solution

- There are $(n-1)!$ permutations of the cities starting at city 1. Check each path takes cn operation. So $c \cdot n!$

$$(1,2,3,4,5,6) \Rightarrow \text{cost} = 19$$
$$(1,2,3,4,6,5) \Rightarrow \text{cost} = 34$$
$$(1,2,3,5,4,6) \Rightarrow \text{cost} = 28$$
$$(1,2,3,5,6,4) \Rightarrow \text{cost} = 15$$
$$(1,2,3,6,4,5) \Rightarrow \text{cost} = 22$$
$$(1,2,3,6,5,4) \Rightarrow \text{cost} = 19$$
$$\dots$$

Better Solutions?

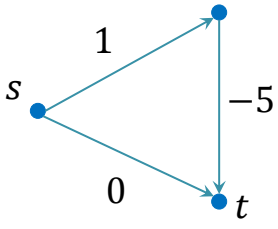
- Using a technique called ***dynamic programming***, one can obtain a running time of $c2^n n^2$.
- Finding a polynomial time algorithm or proving that such an algorithm does not exist would get you **1,000,000$** !

The Millennium Prize Problem

- P versus NP.
- The Hodge conjecture.
- ~~The Poincaré conjecture.~~ *Solved*!
- The Riemann hypothesis.
- Yang–Mills existence and mass gap.
- Navier–Stokes existence and smoothness.
- The Birch and Swinnerton-Dyer conjecture.

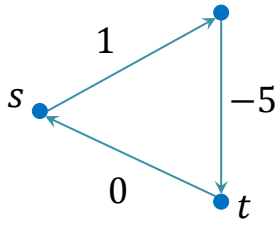
Adding Negative Weights

- Is Dijkstra's algorithm still correct when allowing negative weights?
  - No!



Adding Negative Weights (cont.)

- Is the problem even well defined when allowing negative weights?
  - No! What is the weight of the shortest path from s to t.



The Bellman-Ford Algorithm

- The ***Bellman-Ford algorithm*** receives a weighted directed graph, possibly with negative weights, and a vertex s.
  - If there is a path from s to a negative cycle it reports about it.
  - Otherwise, it finds shortest paths from s to any other vertex.
- We will only use this algorithm as a "black box".

Arbitrage

- **Problem.** Given n types of currency and the conversion rates between them, describe an efficient algorithm that checks whether there exists a series of conversions that starts with 1 unit of a certain currency and ends up with more.

Exchange Rate Table

From/To	USD	AUD	GBP	EUR	INR	JPY	SGD	CHF
USD	1	1.29249	0.55337	0.80425	43.5000	109.920	1.64790	1.24870
AUD	0.77370	1	0.42815	0.62225	33.6560	85.0451	1.27498	0.96612
GBP	1.80710	2.33566	1	1.45335	78.6088	198.636	2.97792	2.25652
EUR	1.24340	1.60708	0.68806	1	54.0879	136.675	2.04900	1.55263
INR	0.02299	0.02971	0.01272	0.01849	1	2.5269	0.03788	0.02871
JPY	0.00910	0.01176	0.00503	0.00732	0.39574	1	0.01499	0.01136
SGD	0.60683	0.78433	0.33581	0.48804	26.3972	66.7031	1	0.75775
CHF	0.80083	1.03507	0.44316	0.64407	34.8362	88.0275	1.31969	1

First Ideas

- Denote by a_{ij} the amount that we obtain by converting 1 unit of currency c_i to currency c_j.
- Build a weighted directed graph:
  - A *vertex* for every type of currency.
  - An *edge* from every vertex to every vertex.
  - The edge from c_i to c_j is of *weight* a_{ij}.
- A cycle $c_{i_1} \to c_{i_2} \to c_{i_3} \to \cdots \to c_{i_1}$ is of weight $a_{i_1 i_2} + a_{i_2 i_3} + \cdots + a_{a_{i_k} i_1}$.
  - But we are looking for $a_{i_1 i_2} \cdot a_{i_2 i_3} \cdots a_{i_k i_1} > 1$.

Addressing the Problem

- We are looking for a cycle $c_1 \to c_2 \to \cdots \to c_k \to c_1$ such that

$$a_{i_1 i_2} \cdot a_{i_2 i_3} \cdots a_{i_k i_1} > 1$$

$$\Updownarrow$$

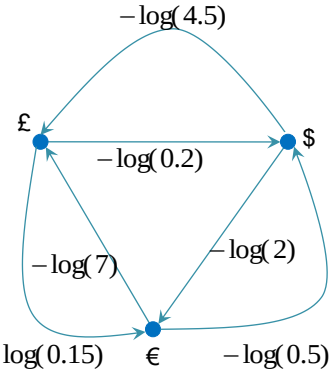
$$\log a_{i_1 i_2} + \log a_{i_2 i_3} + \cdots + \log a_{i_k i_1} > 0$$

$$\Updownarrow$$

$$-\log a_{i_1 i_2} - \log a_{i_2 i_3} - \cdots - \log a_{i_k i_1} < 0$$

Example

	£	$	€
£	1	0.2	0.15
$	4.5	1	2
€	7	0.5	1



The Full Solution

- The algorithm.
  - Build a graph as before, but instead of weights of the form a_{ij}, use $-\log a_{ij}$.
  - Run Bellman-Ford to check whether there is a negative cycle.
  - A negative cycle corresponds to a profitable series of conversions.
  - What is missing? Bellman-Ford requires a vertex to start from.
  - We can choose any vertex because the graph is complete.

The End