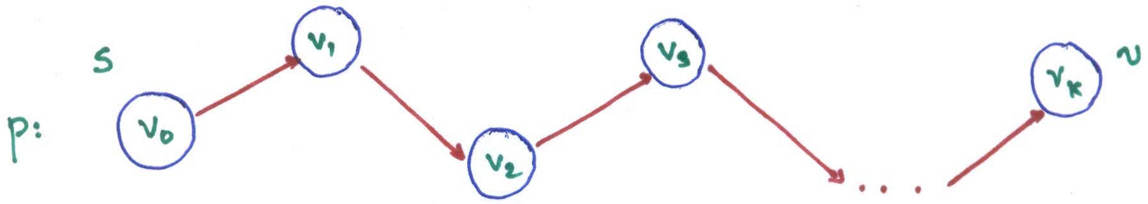
Week 10: Lecture Notes

Topics: Correctness of Bellman Ford
Application of Bellman Ford
All pairs shortest path
Floyd-Warshall
Johnson Algorithm

Correctness of Bellman Ford

Theorem: If $G = (V, E)$ contains no negative-weight cycles then after the Bellman-Ford algorithm executes, $d[v] = \delta(s, v)$ for all $v \in V$

Proof:
Let $v \in V$ be any vertex, and consider a shortest path p from s to v within the minimum number of edges.



Since p is a shortest path, we have
$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$$

Initially, $d[v_0] = 0 = \delta(s, v_0)$, and $d[s]$ is unchanged by subsequent relaxations.

- After 1 pass through E, we have $d[v_1] = \delta(s, v_1)$
- After 2 passes through E, we have $d[v_2] = \delta(s, v_2)$

  ⋮
- After k passes, we have $d[v_k] = \delta(s, v_k)$

Since G contains no negative-weight cycles, p is simple.
Longest simple path has $\leq |V| - 1$ edges.

Detection of negative-weight cycles

Corollary: If a value $d[v]$ fails to converge after $|V|-1$ passes, there exists a negative-weight cycle in G reachable from s.

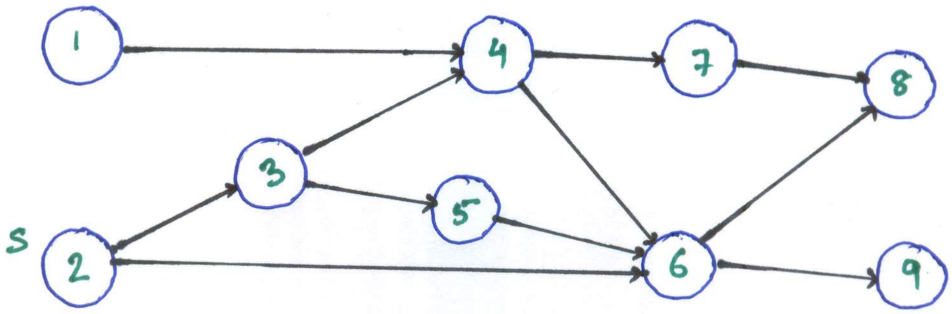
DAG Shortest paths

If the graph is a directed acyclic graph (DAG), we first topollogically sort the vertices.

Determine $f: V \rightarrow \{1, 2, \ldots, |V|\}$ such that $(u,v) \in E$

$\Rightarrow f(u) < f(v)$

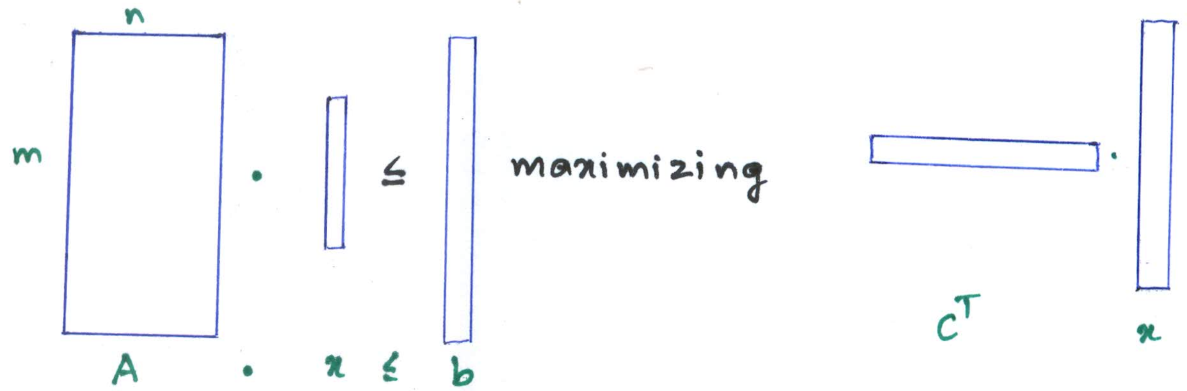
$O(V+E)$ time using depth-first search.



Walk through the vertices $u \in V$ in this order, relaxing the edges in $Adj[u]$, thereby obtaining the shortest paths s in a total of $O(V+E)$ time.

Linear Programming

Let "A" be an $m \times n$ matrix, "b" be an m-vector and "c" be an n-vector. Find an n-vector "x" that maximizes $c^T x$ subject to $Ax \le b$, or determines that no such solution exists.



Linear Programming Algorithms

Algorithms for the general problem

- Simplex methods – practical but worst case exponential time
- Ellipsoid algorithm - polynomial time, but slow in practive
- Interior point methods - polynomial time and competes with simplex

Feasibility problem:

No optimization criterian.

Just find x such that $Ax \le b$.

- In general, just as hard as ordinary LP.

Solving a system of difference constraints

Linear programming where each row of "A" contains exactly one 1, one -1 and the rest 0's.

Example:

$$\left.\begin{array}{l} x_1 - x_2 \le 3 \\ x_2 - x_3 \le -2 \\ x_1 - x_3 \le 2 \end{array}\right\} x_j - x_i \le w_{ij}$$

Solution:

$$x_1 = 3$$
$$x_2 = 0$$
$$x_3 = 2$$

Constraint Graph:

$$x_j - x_i \le w_{ij} \Longrightarrow v_i \xrightarrow{w_{ij}} v_j$$

(The A matrix has dimensions $|E| \times |V|$)

Unsatisfiable Constraints

Theorem: If the constraint graph contains a negative-weight cycle, then the system of differences is unsatisfiable.

Proof:

Suppose that the negative weight cycle is $v_1 \to v_2 \to \dots \to v_k \to v_1$. Then, we have

$$\begin{array}{l} x_2 - x_1 \le w_{12} \\ x_3 - x_2 \le w_{23} \\ \quad\vdots \\ x_k - x_{k-1} \le w_{k-1,k} \\ x_1 - x_k \le w_{k1} \\ \hline 0 \le \text{weight of cycle} \\ \quad < 0. \end{array}$$

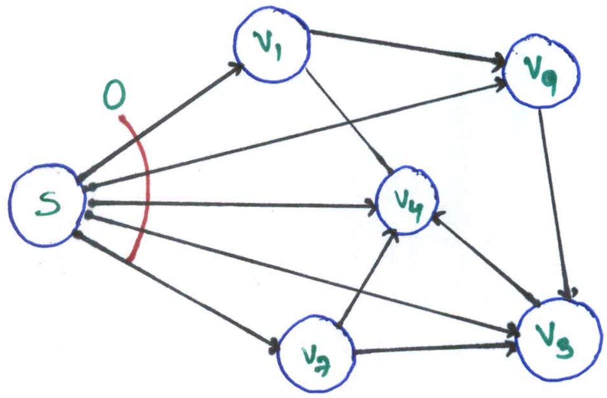
Therefore, no values for x_i can satisfy the constraints.

Satisfying the constraints

Theorem: Suppose no negative weight cycle exists in the constraint graph. Then the constraints are satisfiable.

Proof:

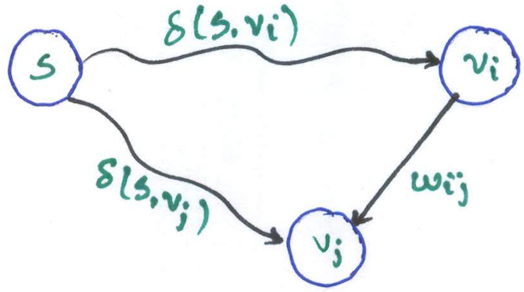
Add a new vertex s to V with a zero weight edge to each vertex $v_i \in V$



Note:
No negative-weight cycles introduced
⇒ shortest paths exist.

Claim:

The assignment $x_i = \delta(s, v_i)$ solves the constraints. Consider any constraint $x_j - x_i \le w_{ij}$ and consider the shortest paths from s to v_j and v_i.



The triangle inequality gives us $\delta(s, v_j) \le \delta(s, v_i) + w_{ij}$. Since $x_i = \delta(s, v_i)$ and $x_j = \delta(s, v_j)$, the constraint $x_j - x_i \le w_{ij}$ is satisfied.

Bellman-Ford and linear programming

Corollary: The Bellman-Ford algorithm can solve a system of m difference constraints on n variables in $O(mn)$ time.

- Single-source shortest paths is a simple LP problem.
- In fact, Bellman-Ford maximizes $x_1 + x_2 + \cdots + x_n$ subject to the constraints $x_j - x_i \le w_{ij}$ and $x_i \le 0$
- Bellman-Ford also minimizes $\max_i\{x_i\} - \min_i\{x_i\}$

Shortest Paths

Single-source shortest paths:

- Non-negative edge weights
  - Dijkstra's algorithm - $O(E + V\log V)$
- General
  - Bellman Ford - $O(VE)$
- DAG
  - One pass of Bellman ford $O(V+E)$

All-pairs shortest paths

- Non-negative edge weights
  - Dijkstra's algorithm $|V|$ times - $O(VE + V^2 \log V)$

All-pairs Shortest Paths

Input: Digraph $G = (V, E)$, where $|V| = n$, with edge-weight function $w: V \to R$

Output: $n \times n$ matrix of shortest-path lengths $\delta(i,j)$ for all $i, j \in V$.

Idea #1.
- Run Bellman Ford once from each vertex
- Time = $O(V^2 E)$.
- Dense graph $\Rightarrow$ $O(V^4)$ time

"Good first try"

Dynamic Programming

Consider the $n \times n$ adjacency matrix $A = (a_{ij})$ of the digraph, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

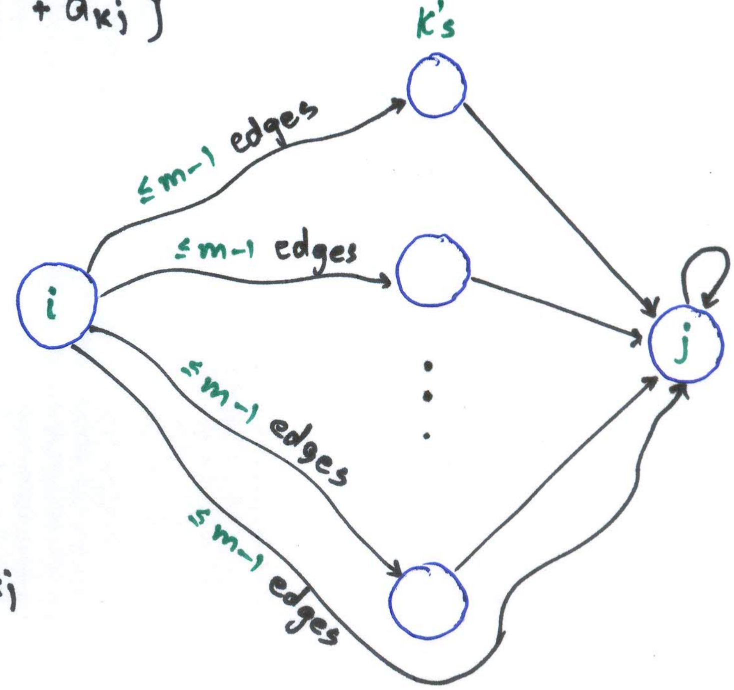
$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$

and for $m = 1, 2, \ldots, n-1$,

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$

Proof of claim:

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$



Relaxation!

for $k \leftarrow 1$ to n
  do if $d_{ij} > d_{ik} + a_{kj}$
    then $d_{ij} \leftarrow d_{ik} + a_{kj}$

Note: No-negative weight cycle implies

$$\delta(i,j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \cdots$$

Matrix Multiplication

Compute $C = A \cdot B$, where C, A and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} .$$

Time = $\Theta(n^3)$ using the standard algorithm.

What if we map "+" → "min" and "·" → "+" ?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}$$

Thus, $D^{(m)} = D^{(m-1)}$ "x" A

Identity matrix = $I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)})$.

The (min, +) multiplication is associative, and with the real numbers, it forms an algebraic structure called a closed semiring.

Consequently, we can compute

$$D^{(1)} = D^{(0)} \cdot A = A^1$$

$$D^{(2)} = D^{(1)} \cdot A = A^2$$

$$\vdots$$

$$D^{(n-1)} = D^{(n-2)} \cdot A = A^{n-1}$$

yielding $D^{(n-1)} = (\delta(i,j))$

Time = $\Theta(n \cdot n^3) = \Theta(n^4)$

No better than $n \times$ B-F.

Improved matrix multiplication algorithm

Repeated squaring:

$$A^{2k} = A^k \times A^k$$

Compute $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\log n)\text{ squarings.}}$

Note:

$$A^{n-1} = A^n = A^{n+1} = \cdots$$

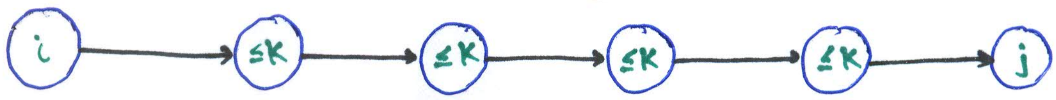
Time $= \Theta(n^3 \lg n)$

To detect negative weight cycles, check the diagonal for negative values in $O(n)$ additional time.

Floyd-Warshall Algorithm

Also dynamic programming, but faster!

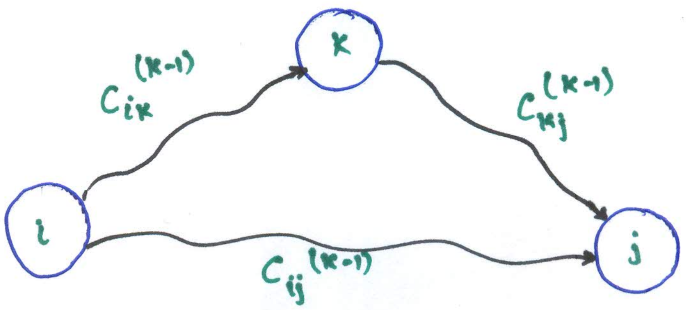
Define $c_{ij}^{(k)}$ = weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$



Thus, $\underline{\delta(i,j) = c_{ij}^{(n)}}$. Also, $\underline{c_{ij}^{(0)} = a_{ij}}$.

Floyd-Warshall Recurrence

$$c_{ij}^{(k)} = \min_k \left\{ c_{ij}^{(k-1)},\ c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \right\}$$



intermediate vertices in $\{1, 2, \ldots, k\}$

Pseudocode for Floyd-Warshall

```
for k ← 1 to n
    do for i ← 1 to n
        do for j ← 1 to n
            do if c_ij > c_ik + c_kj
                then c_ij ← c_ik + c_kj
```

relaxation

Notes:

Okay to omit superscripts, since extra relaxations can't hurt

Runs in $\Theta(n^3)$ time

Simple to code

Efficient in practice.

Transitive Closure of a directed graph

$$\text{Compute } t_{ij} = \begin{cases} 1, & \text{if there exists a path from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases}$$

IDEA:

Use Floyd-Warshall, but with $(\vee, \wedge)$ instead of $(\min, +)$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} \right)$$

Time = $\Theta(n^3)$

Graph reweighting

Theorem:

Given a label $h(v)$ for each $v \in V$, reweight each edge $(u,v) \in E$ by

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

Then all paths between the same two vertices are reweighted by the same amount.

Proof:

Let $p = v_1 \to v_2 \to \dots \to v_k$ be a path in the graph.

Then we have,
$$\begin{aligned}
\hat{w}(p) &= \sum_{i=1}^{k-1} \hat{w}(v_i, v_{i+1}) \\
&= \sum_{i=1}^{k-1} \left(w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) \right) \\
&= w(p) + h(v_k) - h(v_1)
\end{aligned}$$

Johnson's Algorithm

1. Find a vertex labeling h such that $\hat{w}(u,v) \geq 0$ for all $(u,v) \in E$ by using Bellman-ford to solve the difference constraints: $h(v) - h(u) \leq w(u,v)$ or determining that a negative weight cycle exists.
   - Time = $O(VE)$

2. Run Dijkstra's algorithm from each vertex using $\hat{w}$.
   - Time = $O(VE + V^2 \log V)$

3. Reweight each shortest-path length $\hat{w}(p)$ to produce the shortest-path lengths $w(p)$ of the original graph
   - Time = $O(V^2)$

Total time = $O(VE + V^2 \log V)$