Advanced Algorithms

Winter term 2019/20

Lecture 7. Shortest Paths in Graphs with Negative Weights

Steven Chaplick & Alexander Wolff

Chair for Computer Science I
Motivation

Algorithm. $x_n < \cdots < x_2 < x_1 = 0$. Let $n \geq h > v \geq 1$.

for each $hv \in E$: $x_v - x_h < \min\{\ell_h, \ell_v\}$

for each $hv \not\in E$: $x_v - x_h > \min\{\ell_h, \ell_v\}$

Solve LP (without objective function):

# variables $= n-1$  # constraints $= |H| \cdot |V| + n-1$

$\Rightarrow$ runtime: $O(n^{3.5} \log(L+n))$

$H$ $V$

Exercise: Improve this to $O(|E| + n)!$
Solving Systems of Difference Constraints

Is this system feasible?

\[
\begin{align*}
x_1 - x_2 & \leq 0 \\
x_1 - x_5 & \leq -1 \\
x_2 - x_5 & \leq 1 \\
x_3 - x_1 & \leq 5 \\
x_4 - x_1 & \leq 4 \\
x_4 - x_3 & \leq -1 \\
x_5 - x_3 & \leq -3 \\
x_5 - x_4 & \leq -3
\end{align*}
\]

**Definition.** The constraint graph $G_{A,b}$ is a weighted digraph with vertex set $V_A = \{v_0, v_1, \ldots, v_n\}$ and edge set $E_A = \{v_i v_j : x_j - x_i \leq b_{ij} \text{ is a constraint}\} \cup \{v_0 v_k : 1 \leq k \leq n\}$. The weight of $v_i v_j$ is $b_{ij}$ if $i > 0$ and 0 otherwise.
Shortest Paths Do the Job

**Theorem.** Let $Ax \leq b$ be a system of difference constraints, and let $\delta_k = \delta(v_0, v_k)$ be the length of a shortest $v_0 - v_k$ path for $k = 1, \ldots, n$. If $G_{A,b}$ contains no negative cycles, then $x = (\delta_1, \ldots, \delta_n)$ is a feasible solution. If $G_{A,b}$ contains a negative cycle, then there is no feasible solution.

**Proof.** Assume no neg. cycles. Consider $v_i v_j \in E_A$ with $i > 0$. \(\Delta\)-inequality $\Rightarrow$ $\delta_j \leq \delta_i + b_{ij}$, or $\delta_j - \delta_i \leq b_{ij}$. Letting $x_i = \delta_i$ and $x_j = \delta_j$ satisfies $x_j - x_i \leq b_{ij}$.

Now assume $\exists$ neg. cycle and $Ax \leq b$ has a solution $x$. Wlog., let $C = \langle v_1, v_2, \ldots, v_k = v_1 \rangle$ be a neg. cycle. $\Rightarrow x_2 - x_1 \leq b_{12}$, $x_3 - x_2 \leq b_{23}$, $\ldots$, $x_k - x_{k-1} \leq b_{k-1,k}$. $\Rightarrow 0 \leq b_{12} + b_{23} + \cdots + b_{k-1,k} = w(C)$ $\blacksquare$
Shortest Paths & Negative Edge Weights

**Ideas?**

Dijkstra can handle only graphs with non-negative edge weights.

But maybe we can reduce to this problem?

What about adding the same constant $c$ to each edge weight?

Problem: Paths with few edges get relatively cheaper :-(

Recall initialization and main subroutine of Dijkstra:

\[
\text{Initialize}(\text{graph } G, \text{ vtx } s) \\
\text{foreach } u \in V \text{ do} \\
\quad u.d = \infty \quad \text{estimate for } \delta(s, u) \\
\quad u.\pi = \text{nil} \quad \text{pointer to predecessor on some “currently” shortest } s-u \text{ path} \\
\quad s.d = 0
\]

\[
\text{Relax(vtx } u, \text{ vtx } v, \text{ weights } w) \\
\quad \text{if } v.d > u.d + w(u, v) \text{ then} \\
\quad \quad v.d = u.d + w(u, v) \\
\quad \quad v.\pi = u \quad [Q.\text{DecreaseKey}(v, v.d)]
\]
The Bellman–Ford Algorithm

Dijkstra(graph G, weights w, vtx s)
Initialize(G, s)
Q = new PriorityQueue(G.V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    foreach v ∈ Adj[u] do
        Relax(u, v; w)

Bellman–Ford(graph G, weights w, vtx s)
Initialize(G, s)
for i = 1 to |G.V| – 1 do
    foreach uv ∈ G.E do
        Relax(u, v; w)
    foreach uv ∈ G.E do
        if v.d > u.d + w(u, v) return false
return true

runtime?
O(E + V log V)

runtime?
O(V · E)
Correctness of Bellman–Ford

If $G$ contains no neg. cycle reachable from $s$, after the for-$i$ loop, for every vertex $v$, $v.d = \delta(s, v)$ and Bellman–Ford returns true.

Suppose $v$ is reachable from $s$.

\[ \Rightarrow \exists \text{ shortest } s\text{–}v \text{ path } \delta = \langle s = v_0, v_1, \ldots, v_k = v \rangle. \]

$G$ has no negative cycle, $\delta$ shortest path $\Rightarrow$ length $k \leq n - 1$. After initialization, $v_0.d = \delta(s, v_0) = 0$.

In phase $i$ of the alg., $v_{i-1}v_i$ is relaxed.

\[ \Rightarrow v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i). \text{ By induction, } v_{i-1}.d = \delta(s, v_{i-1}). \]

\[ \Rightarrow v_i.d \leq \delta(s, v_{i-1}) + w(v_{i-1}, v_i) = \delta(s, v_i) \overset{\delta \text{ s.p.}}{\Rightarrow} v_i.d = \delta(s, v_i). \]

Suppose $v$ is *not* reachable from $s$. $\Rightarrow$ Initially, $v.d = \infty$

$\Rightarrow$ During execution, $v.d$ remains $\infty$ (otherwise $\exists$ $s$–$v$ path)

$\Rightarrow$ At termination, $v.d = \infty = \delta(s, v)$. $\Box$
The Bellman–Ford Algorithm (overview)

Initialize(graph $G$, vtx $s$)

- foreach $u \in V$ do
  - $u.d = \infty$
  - $u.\pi = \text{nil}$
- $s.d = 0$

Relax(vtx $u$, vtx $v$, weights $w$)

- if $v.d > u.d + w(u, v)$ then
  - $v.d = u.d + w(u, v)$
  - $v.\pi = u$

Bellman–Ford(graph $G$, weights $w$, vtx $s$)

- Initialize($G$, $s$)
- for $i = 1$ to $|G.V| - 1$ do
  - foreach $uv \in G.E$ do
    - Relax($u$, $v$; $w$)
- foreach $uv \in G.E$ do
  - if $v.d > u.d + w(u, v)$ return false
- return true
Correctness (cont’d)

If $G$ contains a negative cycle that is reachable from $s$, then Bellman–Ford returns false.

Let $C = \langle v_0, v_1, \ldots, v_k = v_0 \rangle$ be such a negative cycle.

Assume that Bellman-Ford returns true.

$\Rightarrow v_1.d \leq v_0.d + w(v_0, v_1), \ldots, v_k.d \leq v_{k-1}.d + w(v_{k-1}, v_k),$

$\Rightarrow 0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i) = w(C)$

For this implication we additionally need that $\sum_i v_i.d < \infty$.

(True since $C$ is reachable from $s$, plus the previous proof.)

Improvement: $O(\sqrt{VE} \log W)$, where $W = \max_{uv \in E} w(u, v)$.

All-Pairs Shortest Paths

Assume that the graph is given by a matrix $W = (w_{ij})_{1 \leq i,j \leq n}$. Let $\ell_{ij}^{(m)}$ be the length of a shortest $i$–$j$ path with $\leq m$ edges.

Then $\ell_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{otherwise} \end{cases}$ and $\ell_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ \ell_{ik}^{(m-1)} + w_{kj} \}$.

$\Rightarrow \delta(i, j) = \ell_{ij}^{(n-1)} = \ell_{ij}^{(n)} = \ell_{ij}^{(n+1)} = \ldots$ since shortest paths are simple (if there are no neg. cycles)!

Extend-Shortest-Paths($L$, $W$)

$L' = (\ell'_{ij} = \infty)$ new $n \times n$ matrix

for $i = 1$ to $n$ do

\[ \text{for } j = 1 \text{ to } n \text{ do} \]

\[ \text{for } k = 1 \text{ to } n \text{ do} \]

\[ \ell'_{ij} = \min \{ \ell'_{ij}, \ell_{ik} + w_{kj} \} \]

return $L'$

Slow-All-Pairs-SP($W$)

$L^{(1)} = W$

for $m = 2$ to $n - 1$ do

$L^{(m)} = \text{new matrix}$

$L^{(m)} = \text{ESP}(L^{(m-1)}, W)$

return $L^{(n-1)}$

Runtime: $O(n^4)$
Faster APSP

Faster-All-Pairs-SP($n \times n$ matrix $W$)

$L^{(1)} = W$

$m = 1$

while $m < n - 1$ do

$L^{(2m)} =$ new $n \times n$ matrix

$L^{(2m)} =$ Extend-Shortest-Path($L^{(m)}$, $L^{(m)}$)

$m = 2m$

return $L^{(m)}$

Runtime: $O(n^3 \log n)$
The Floyd–Warshall Algorithm

Floyd–Warshall\((n \times n\) matrix \(W\))

\[
D^{(0)} = W
\]

for \(k = 1\) to \(n\) do

\[
D^{(k)} = \text{new } n \times n\text{ matrix}
\]

for \(i = 1\) to \(n\) do

for \(j = 1\) to \(n\) do

\[
d^{(k)}_{ij} = \min\{d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj}\}
\]

return \(D^{(n)}\)

\[d^{(0)}_{ij} = w_{ij}; \text{ if } k > 0 \text{ then}
\]

\[d^{(k)}_{ij} = \min\{d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj}\}\]

Intermediate vertices \(\leq k\)

Vertices \(< k\)

Vertices \(< k\)

Floyd–Warshall's algorithm [J. ACM 1977]

Improvement:

\[O(V(V \log V + E))\]

Johnson's algorithm [J. ACM 1977]

Runtime:

\[O(n^3)\]