Title

Ferienakademie im Sarntal — Course 2
Distance Problems: Theory and Praxis

Nesrine Damak

Fakultät für Informatik
TU München

20. September 2010

Outline

Definition

A graph $G(V, E)$ is a set V of vertices, a set E of edges, and a real-valued weight function $w : E \longrightarrow R$.

Definition

A path P in G is a sequence of vertices $(v_1, .., v_n)$ such that $(v_i, v_{i+1}) \in E$ for all $1 \leq i < n$ and $v \in V$.

Definition

The weight of a path P in G is the sum of the weights of its edges: $w(P) = \sum_i w(v_{i-1}, v_i)$.

Definition

For nodes $u, v \in V$, the shortest path weight from u to v is defined to be: $\delta(u, v) = min(w(P))$ if such a P exists or infinity otherweise.

Definition

For nodes $u, v \in V$ a shortest path from u to v is a path with $w(P) = \delta(u, v)$

Applications of the Shortest Path Problem

- find the best route to drive between Berlin and Munich or figure how to direct packets to a destination across a network
- image segmentation
- speech recognition
- find the center point of a graph: the vertex that minimizes the maximum distance to any other vertex in the graph.

Input: A graph G =(V,E), an edge weight function w and a node $s \in V$.
Output: A shortest path P from s to all other vertices $v \in V - \{s\}$.
Algorithmen:

- unweighted case : BFS
- no negativ edge-weights : Dijkstra-Algorithm
- general case : Bellman-Ford Algorithm

Notation

- $dist(x)$: distance from the initial node to x, the shortest path found by the algorithm so far
- Q: FIFO queue

- Suppose that $w(u, v) = 1$ for all $uv \in E$
- We use a simple FIFO queue
- Analysis Time: $O(|V| + |E|)$

BFS(G)

while $Q \neq empty$ **do**
    $u \leftarrow DEQUEUE(Q)$;
    for $each\ v \in adj(u)$ **do**
        if $dist(v) = \infty$ **then**
            $dist(v) = dist(u) + 1$;
            $ENQUEUE(Q, v)$;
        end
    end
end

BFS is used to solve following problems:

- Testing whether graph is connected.
- Computing a spanning forest of graph.
- Computing, for every vertex in graph, a path with the minimum number of edges between start vertex and current vertex or reporting that no such path exists.
- Computing a cycle in graph or reporting that no such cycle exists.

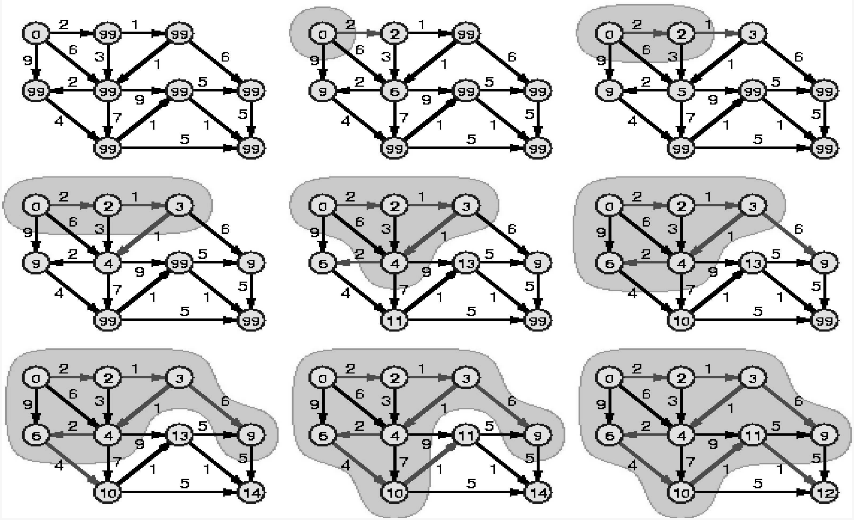
Notation

- S:a set of vertices, whose shortest path distances from s are known ("the solved set").
- $dist(x)$: distance from the initial node to x
- Q: priority queue maintaining V-S
- pred(v): the predecessor of the vertex v

Dijkstra(G,w,s)

dist(s) $\leftarrow$ 0;
for $v \in V - \{s\}$ **do**
    dist(v)$\leftarrow \infty$;
end
$S \leftarrow empty$;
$Q \leftarrow V$;
while $Q \neq empty$ **do**
    $u \leftarrow Extract - Min(Q)$;
    $S \leftarrow S \cup \{u\}$;
    for $v \in adj(u)$ **do**
        if $dist(v) > dist(u) + w(u, v)$ **then**
            $dist(v) \leftarrow dist(u) + w(u, v)$;
        end
    end
end

DIJKSTRA'S ALGORITHM



Proof of correctness

Theorem

Dijkstras Algorithm terminates with $dist(v) = \delta(s, v)$ *for all* $v \in V$.

Proof.

We show: $dist(v) = \delta(s, v)$ for every $v \in V$ when v is added to S.

Supposition:

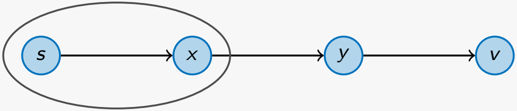
- v is the first vertex added to S , so that $dist(v) > \delta(s, v)$
- y is the first vertex in $V - S$ along a shortest path from s to v
- $x = pred(y) \in S$

then:

- $dist(x) = \delta(s, x)$
- xy is relaxed: $dist(y) = \delta(s, y) \leq \delta(s, v) < dist(v)$

CONTRADICTION because $dist(v) \leq dist(y)$!

$\square$


s
x
y
v

Time complexity

Supposition: Fibonacci-heaps

Initialization	$O(\|V\|)$
Extract-Min	$\|V\|O(log\|V\|)$
DecreaseKey	$\|E\|O(\|V\|)$

so is the time complexity of Dijkstra :$O(|E| + |V|log|V|)$

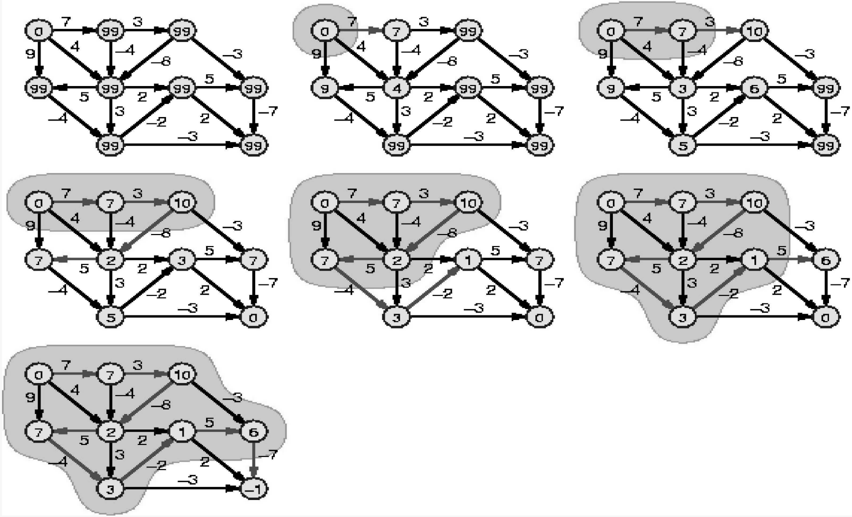
The main idea is that if the edge uv is the last edge of the shortest path to v, the cost of the shortest path to v is the cost of the shortest path to u plus the weight of w(u,v).
Bellman-Ford algorithm finds all shortest-path lengths from a source $s \in V$ to all $v \in V$ or determines that a negative-weight cycle exists.

Bellman-Ford(G,w,s)

dist(s) $\leftarrow$ 0;
for $v \in V - \{s\}$ **do**
    dist(v)$\leftarrow \infty$;
    $pred(v) := null$;
end

```
for i from 1 to |V| − 1 do
    for each edge uv ∈ E do
        if dist(u) + w(u, v) < dist(v) then
            dist(v) := dist(u) + w(u, v) ;
            pred(v) := u ;
        end
    end
end
for each edge uv ∈ E do
    if dist(u) + w(u, v) < dist(v) then
        error "Graph contains a negative-weight cycle" ;
    end
end
```

BELLMAN–FORD ALGORITHM



Proof of correctness

Theorem

if $G = (V, E)$ contains no negative-weight cycles, then after the Bellman-Ford algorithm executes, $dist(v) = \delta(s, v)$ for all $v \in V$

Proof.

let $v \in V$ be any vertex, and consider a shortest path P from s to v with the minimum number of edges.

we have: $\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$.

Initially: $dist(v_0) = 0 = \delta(s, v_0)$

After 1 pass through E : $dist(v_1) = \delta(s, v_1)$

.

.

After k passes through E : $dist(v_k) = \delta(s, v_k)$

Longest simple path has $\leq |V| - 1$ edges.

□

Time complexity

The Bellman-Ford algorithm simply relaxes all the edges, and does this $|V| - 1$ times: It runs in $O(|V| \cdot |E|)$ time: the best time bound for the sssp problem.

Input: A connected graph G=(V,E) and an edge weight function w.
Output: For all pairs $u, v \in V$ of nodes a shortest path from u to v
$\rightarrow$ a $|V| \times |V|$ -matrix.
Possible algorithms:

- Naive implementation: Use standard single-source algorithms $|V|$ times
  - Dijkstra : running a $O(E + VlogV)$ process $|V|$ times
  - Bellman–Ford algorithm on a dense graph will take about $O(V^2E)$
- Floyd Warshall algorithm
- Johnson algorithm

Notation

- $dist^{(k)}(i,j)$ is the distance from i to j with intermediate vertices belonging to the set $\{1, 2, ..., k\}$

Floyd-Warshall(G,w)

```
for i = 1 to V do
    for j = 1 to V do
        if there is an edge from i to j then
            dist^(0)(i,j) = the length of the edge from i to j ;
        end
        dist^(0)(i,j) = ∞ ;
    end
end
for k = 1 to V do
    for i = 1 to V do
        for j = 1 to V do
            dist^(k)(i,j) =
            min(dist^(k−1)(i,j), dist^(k−1)(i,k) + dist^(k−1)(k,j)) ;
        end
    end
end
```

- The algorithms running time is clearly $O(|V|^3)$
- $DIST^{(k)}$ is the $|V| \times |V|$ Matrix $[dist^{(k)}(i,j)]$.
- $dist^{(0)}(i,j) = w(i,j)$ (no intermediate vertex $=$ the edge from i to j)
- Claim: $dist^{(|V|)}(i,j)$ is the length of the shortest path from i to j . So our aim is to compute $DIST^{(|V|)}$.
- Subproblems: compute $DIST^{(k)}$ for $k = 0, ..., |V|$ (dynamic!)

How do we compute $dist^{(k)}(i,j)$ assuming that we have already computed the previous matrix $DIST^{(k-1)}$?
We dont go through k at all: Then the shortest path from i to j uses only intermediate vertices $\{1,...,k-1\}$ and hence the length of the shortest path is $dist^{(k-1)}(i,j)$.

We do go through k: we can assume that we pass through k exactly once (no negative cycles). That is, we go from i to k, and then from k to j: we should take the shortest path from i to k, and the shortest path from k to j (optimality).
Each of these paths uses intermediate vertices only in $\{1, 2, .., k-1\}$. The length of the path is : $dist^{(k-1)}(i,k) + dist^{(k-1)}(k,j)$.
This suggests the following recursive rule for computing $DIST^{(k)}$:

- $dist^{(0)}(i,j) = w(i,j)$
- $dist^{(k)}(i,j) = min(dist^{(k-1)}(i,j), dist^{(k-1)}(i,k) + dist^{(k-1)}(k,j))$

After $|V|$ iterations, $dist^{|V|}(i,j)$ is the shortest path between i and j.

Notation

a shortest-path tree rooted at the source vertex s is a directed subgraph $G' = (V', E')$ where V' and E' are subsets of V and E respectively, such that

- V' is the set of vertices reachable from s in G
- G' forms a rooted tree with root s
- for all $v \in V'$, the unique path from s to $v \in G'$ is the shortest path from s to v in G

Idea:

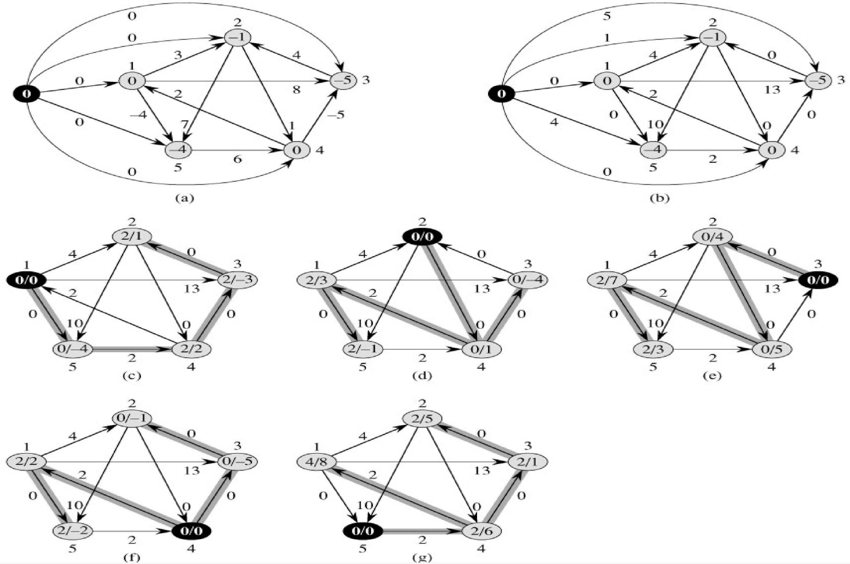
- Add a new node s so that $w(s, v) = 0$ for all $v \in V \rightarrow$ a new graph G'.
- Use the BellmanFord algorithm to check for negative weight cycles and find $h(v) = \delta(s, v)$ in G'.
- Reweight the edges using $h(v)$ with the reweighting function $\hat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$.
- Use Dijkstras algorithm on the transformed graph (with no negative edges) in order to find the shortest path.

Pseudocode

Johnson(G,w)

Compute G', where $V[G'] = V[G] \cup s$;
$E[G'] = E[G] \cup (s, v) : v \in V[G]$;
for *all* $v \in V[G]$ **do**
  $w(s, v) = 0$;
end

```
if BELLMAN − FORD(G′, w, s) = FALSE then
    print the input graph contains a negative weight cycle ;
end
for each vertex v ∈ V[G′] do
    set h(v) to the value of δ(s, v) computed by the Bellman-Ford alg.;
end
for each edge (u, v) ∈ E[G′] do
    ŵ(u, v) ← w(u, v) + h(u) − h(v) ;
    for each vertex u ∈ V[G] do
        run DIJKSTRA(G, ŵ, u) to compute δ′(u, v) for all v ∈ V[G];
        for each vertex v ∈ V[G] do
            dist(u, v) ← δ′(u, v) + h(v) − h(u) ;
        end
    end
end
```


(a)
(b)
(c)
(d)
(e)
(f)
(g)

Are all the $\hat{w}$'s non-negative? YES

$\delta(s,u) + w(u,v) \geq \delta(s,v)$ otherwise $s \rightarrow u \rightarrow v$ would be shorter than the shortest path $s \rightarrow v$

Does the reweighting preserve the shortest path? YES

$\hat{w}(p) = \sum \hat{w}(v_i, v_{i+1})$
$= w(v_1, v_2) + \delta(s, v_1) - \delta(s, v_2) + ... + w(v_{k-1}, v_k) + \delta(s, v_{k-1}) - \delta(s, v_k)$
$= w(p) + \delta(s, v_1) - \delta(s, v_k)$

Time complexity

1. Computing G: $\Theta(V)$
2. Bellman-Ford: $\Theta(VE)$
3. Reweighting: $\Theta(E)$
4. Running (Modified) Dijkstra: $\Theta(V^2 lgV + VE)$
5. Adjusting distances: $\Theta(V^2)$

Total is dominated by Dijkstra: $\Theta(V^2 lgV + VE)$

Thank you!