Lecture overview

Shortest paths IV

– review and analysis of Djikstra
– speeding it up
  • faster special cases/implementation
  • one source-one target
    – bidirectional
    – goal-directed searches
Dijkstra’s algorithm

\[d[s] \leftarrow 0\]
for each \(v \in V - \{s\}\)
    do \(d[v] \leftarrow \infty\)
\[S \leftarrow \emptyset\]
\[Q \leftarrow V\]
while \(Q \neq \emptyset\)
    do \(u \leftarrow \text{Extract-Min}(Q)\)
        \(S \leftarrow S \cup \{u\}\)
    for each \(v \in \text{Adj}[u]\)
        do if \(d[v] > d[u] + w(u, v)\)
            then \(d[v] \leftarrow d[u] + w(u, v)\)
\{(Implicit \ DECREASE-KEY)\}

initialization

relaxation steps
Correctness — main argument

**Theorem.** Dijkstra’s algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

**Proof.**

It suffices to show that $d[v] = \delta(s, v)$ for every $v \in V$ when $v$ is added to $S$.

• Suppose $u$ is the first vertex added to $S$ for which $d[u] \neq \delta(s, u)$. Let $y$ be the first vertex in $V - S$ along a shortest path from $s$ to $u$, and let $x$ be its predecessor:

$S$, just before adding $u$. 
Analysis of Dijkstra

\[ \text{while } Q \neq \emptyset \]
\[ \text{do } u \leftarrow \text{EXTRACT-MIN}(Q) \]
\[ S \leftarrow S \cup \{u\} \]
\[ \text{for each } v \in \text{Adj}[u] \]
\[ \text{do if } d[v] > d[u] + w(u, v) \]
\[ \text{then } d[v] \leftarrow d[u] + w(u, v) \]

\[ \text{DECREASE-KEY} \]

Time = $\Theta(n) \cdot T_{\text{EXTRACT-MIN}} + \Theta(m) \cdot T_{\text{DECREASE-KEY}}$
Analysis of Dijkstra (continued)

Time = $\Theta(n) \cdot T_{\text{EXTRACT-MIN}} + \Theta(m) \cdot T_{\text{DECREASE-KEY}}$

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$T_{\text{EXTRACT-MIN}}$</th>
<th>$T_{\text{DECREASE-KEY}}$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>binary heap</td>
<td>$O(\lg n)$</td>
<td>$O(\lg n)$</td>
<td>$O(m \lg n)$</td>
</tr>
<tr>
<td>Fibonacci heap</td>
<td>$O(\lg n)$</td>
<td>$O(1)$</td>
<td>$O(n \lg n + m)$</td>
</tr>
</tbody>
</table>

amortized

not covered in 6.006
Special cases of edge weights

- if edge weights are integral and bounded by a small constant $C$
- use an array of lists ($nC+1$ “buckets”) for implementing the priority queue $Q$

- $T_{\text{EXTRACT-MIN}}$ and $T_{\text{DECREASE-KEY}}$ take $O(1)$
- $Time\_Total$ is then $O(n+m)$
**Single-source** \( s \), **Single-target** \( t \), **S.P.P.**

\[
d[s] \leftarrow 0
\]

for each \( v \in V - \{s\} \)
do \( d[v] \leftarrow \infty \)

\( S \leftarrow \emptyset \)
\( Q \leftarrow V \)
while \( Q \neq \emptyset \)
do \( u \leftarrow \text{Extract-Min}(Q) \)
\( S \leftarrow S \cup \{u\} \)
for each \( v \in Adj[u] \)
do if \( d[v] > d[u] + w(u, v) \)
then \( d[v] \leftarrow d[u] + w(u, v) \)

**initialization**

stop whenever \( u = t \) !!

**relaxation steps**
Bi-directional Search

forward search

backward search
Bi-directional Djikstra

- Alternate forward search from $s$, backward search from $t$ (follow edges backward)
- $d_f(u)$ distances for forward search; $d_b(u)$ distances for backward search
- Algorithm terminates when some vertex $w$ has been processed, i.e., deleted from the queue of both searches, $Q_f$ and $Q_b$
Bi-directional Dijkstra, example

**Forward**
- $d_f(s) = 0$
- $d_f(w) = 5$
- $d_f(u) = 3$
- $d_f(u') = 6$

**Backward**
- $d_f(s) = 0$
- $d_f(u) = 3$
- $d_f(u') = 6$
- $d_f(t) = 10$

**Deleted from both queues**

so terminate!
Bi-directional Dijkstra, example

Forward:
- $d_f(s) = 0$
- $d_f(u) = 3$
- $d_f(w) = 5$
- $u'$ not in queue
- $d_f(u') = 6$
- $s$ deleted from Forward queue

Backward:
- $d_f(s) = 0$
- $d_f(u) = 3$
- $d_f(w) = 5$
- $u'$ not in queue
- $d_f(u') = 6$
- $s$ deleted from Backward queue

Common nodes:
- $u$
- $u'$
- $t$
- $w$

Deleted:
- $s$ from Forward
- $s$ from Backward
- $u'$ from Forward
- $u'$ from Backward

Deleted from both queues:
- $s$
- $u'$

Solution:
- $d_f(t) = 10$
- $d_f(w) = 5$
- $d_f(u') = 6$
- $d_f(u) = 3$
- $d_f(s) = 0$
Bi-directional Dijkstra, example

For the forward search:
- \( d_f(s) = 0 \)
- \( d_f(u) = 3 \)
- \( d_f(w) = 5 \)
- \( d_f(t) = 10 \)

For the backward search:
- \( d_b(t) = 0 \)
- \( d_b(u) = 6 \)
- \( d_b(w) = 5 \)
- \( d_b(s) = 10 \)

\( d_f(s) = 0 \) is deleted from the forward queue.
\( d_b(s) = 10 \) is deleted from the backward queue.

Deleting these from both queues causes the algorithm to terminate.

For the forward search:
- \( d_f(u) = 3 \)
- \( d_f(w) = 5 \)
- \( d_f(t) = 10 \)

For the backward search:
- \( d_b(u) = 6 \)
- \( d_b(w) = 5 \)
- \( d_b(t) = 0 \)

\( d_f(u) = 3 \) is deleted from the forward queue.
\( d_b(u) = 6 \) is deleted from the backward queue.

Deleting these from both queues causes the algorithm to terminate.
Goal-directed search or A*

Idea: use a “potential function” $\lambda_t(u)$ over vertices to make the target vertex $t$ “more attractive”

Implementation: Modify edge weights as follow:
$$w^*(u,v) = w(u,v) - \lambda_t(u) + \lambda_t(v)$$
Goal directed search or A*, cont.

Modify edge weights with potential function over vertices:

\[ w^*(u,v) = w(u,v) - \lambda_t(u) + \lambda_t(v) \]

⇒ for any path \( p \) between \( s \) and \( t \) we have:

\[ w^*(p) = w(p) - \lambda_t(s) + \lambda_t(t) \]

⇒ a path from \( s \) to \( t \) is a shortest path under \( w^* \) iff it is a shortest path under \( w \)
Feasible potential and example

• Potential $\lambda_t(u)$ is feasible if
  $$w^*(u,v) = w(u,v) - \lambda_t(u) + \lambda_t(v) \geq 0$$
• As a result can use Dijkstra with $w^*$

• Examples:
  – Euclidean plane
  – Landmarks