Shortest Paths in Directed Planar Graphs with Negative Lengths: a Linear-Space $O(n \log^2 n)$-Time Algorithm

Shay Mozes (Brown University)

joint work with

Philip Klein (Brown University)

Oren Weimann (MIT)
Single-Source shortest paths

- Planar graph
- Directed
- Positive and negative lengths
- No negative cycles
Single-Source shortest paths

- Planar graph
- Directed
- Positive and negative lengths
- No negative cycles
Single-Source shortest paths

- Planar graph
- Directed
- Positive and negative lengths
- No negative cycles
Applications

- Feasible circulation
- Feasible flow
- Perfect matching
- Image segmentation
- Stereo matching
Related Work

General graphs:
- Dijkstra (non-negative lengths) - $O(n \log n + m)$
- Bellman-Ford - $O(nm)$

Planar graphs:
- $O(n^{3/2})$ - [Lipton, Rose and Tarjan 1979]
- $O(n^{4/3} \log^{2/3} D)$ - [Henzinger, Klein, Rao, Subramanian 1994]
  also, $O(n)$ for non-negative lengths
- $O(n \log^3 n)$ time $O(n \log n)$ space - [Fakcharoenphol and Rao 2001]

Our Contribution:
- $O(n \log^2 n)$ time, $O(n)$ space
Rerooting
• We want distances from $s$
• It suffices to find distances from any node $r$
Price Functions, Reduced Lengths

- **Price function**: $\varphi(v)$
- **Reduced length**: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
Price Functions, Reduced Lengths

• Price function: \( \phi(v) \)
• Reduced length: \( l_\phi(uv) = \phi(u) + l(uv) - \phi(v) \)
Price Functions, Reduced Lengths

- **Price function**: \( \varphi(v) \)
- **Reduced length**: \( l_{\varphi}(uv) = \varphi(u) + l(uv) - \varphi(v) \)

![Diagram with nodes and arrows representing the price function and reduced length calculation]
Price Functions, Reduced Lengths

- **Price function:** $\varphi(v)$
- **Reduced length:** $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
Price Functions, Reduced Lengths

- **Price function:** $\phi(v)$
- **Reduced length:** $l_\phi(uv) = \phi(u) + l(uv) - \phi(v)$
Price Functions, Reduced Lengths

- **Price function**: $\phi(v)$
- **Reduced length**: $l_\phi(uv) = \phi(u) + l(uv) - \phi(v)$
Price Functions, Reduced Lengths

- **Price function**: $\varphi(v)$
- **Reduced length**: $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
Price Functions, Reduced Lengths

- **Price function**: \( \varphi(v) \)
- **Reduced length**: \( l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v) \)

- Length of any s-t path changes by \( \varphi(s) - \varphi(t) \)
Price Functions, Reduced Lengths

- **Price function**: $\phi(v)$
- **Reduced length**: $l_\phi(uv) = \phi(u) + l(uv) - \phi(v)$

- Length of any s-t path changes by $\phi(s) - \phi(t)$
- $\phi(v)$ preserves shortest paths
Price Functions, Reduced Lengths

- **Price function:** $\varphi(v)$
- **Reduced length:** $l_\varphi(uv) = \varphi(u) + l(uv) - \varphi(v)$
- Length of any s-t path changes by $\varphi(s)-\varphi(t)$
- $\varphi(v)$ preserves shortest paths
Price Functions, Reduced Lengths

- **Price function**: $\phi(v)$
- **Reduced length**: $l_\phi(uv) = \phi(u) + l(uv) - \phi(v)$
- Length of any s-t path changes by $\phi(s) - \phi(t)$
- $\phi(v)$ preserves shortest paths
Price Functions, Reduced Lengths

- **Price function:** $\phi(v)$
- **Reduced length:** $l_\phi(uv) = \phi(u) + l(uv) - \phi(v)$
- Length of any s-t path changes by $\phi(s) - \phi(t)$
- $\phi(v)$ preserves shortest paths

- Price function is **feasible** if $l_\phi$ is non-negative
- Converts a problem with negative lengths to non-negative lengths
Price Functions, Reduced Lengths

- **Price function**: \( \varphi(v) \)
- **Reduced length**: \( l_{\varphi}(uv) = \varphi(u) + l(uv) - \varphi(v) \)
- Length of any s-t path changes by \( \varphi(s) - \varphi(t) \)
- \( \varphi(v) \) preserves shortest paths

- Price function is **feasible** if \( l_{\varphi} \) is non-negative
- Converts a problem with negative lengths to non-negative lengths
- Single source distances form a feasible price function because \( \varphi(u) + l(uv) \geq \varphi(v) \)
• We want distances from $s$
• It suffices to find distances from any node $r$
Rerooting

- We want distances from $s$
- It suffices to find distances from any node $r$
  - Use distances from $r$ as a feasible price function
Rerooting

• We want distances from $s$
• It suffices to find distances from any node $r$
  
  - Use distances from $r$ as a feasible price function
  - Run Dijkstra’s algorithm from $s$
I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion

II. boundary to boundary distances in $G_i$

III. $r$-to-boundary distances in $G$

IV. distances from $r$ in $G$

V. rerooting
I. recursion

II. boundary to boundary distances in $G_i$

III. r-to-boundary distances in $G$

IV. distances from r in $G$

V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in G
IV. distances from r in G
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion

II. boundary to boundary distances in \( G_i \)

III. r-to-boundary distances in \( G \)

IV. distances from \( r \) in \( G \)

V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in G
IV. distances from r in G
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from r in $G$
V. rerooting
High-Level View

I. recursion

II. boundary to boundary distances in $G_i$

III. r-to-boundary distances in $G$

IV. distances from r in $G$

V. rerooting
High-Level View

I. recursion

II. boundary to boundary distances in $G_i$

III. $r$-to-boundary distances in $G$

IV. distances from $r$ in $G$

V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from r in $G$
V. rerooting
I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion

II. boundary to boundary distances in $G_i$

III. r-to-boundary distances in $G$

IV. distances from r in $G$

V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion

II. boundary to boundary distances in $G_i$

III. $r$-to-boundary distances in $G$

IV. distances from $r$ in $G$

V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. r-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
High-Level View

I. recursion
II. boundary to boundary distances in $G_i$
III. $r$-to-boundary distances in $G$
IV. distances from $r$ in $G$
V. rerooting
I. Recursive Step

Planar Separator:

- $O(\sqrt{n})$ boundary nodes
- At most $2n/3$ nodes in each part
- Can be found in $O(n)$ time

[Lipton-Tarjan 79, Miller 86]
I. Recursive Step

Planar Separator:

- \( O(\sqrt{n}) \) boundary nodes
- At most \( 2n/3 \) nodes in each part
- Can be found in \( O(n) \) time

[Lipton-Tarjan 79, Miller 86]
I. Recursive Step

Planar Separator:

- $O(\sqrt{n})$ boundary nodes
- At most $2n/3$ nodes in each part
- Can be found in $O(n)$ time
  [Lipton-Tarjan 79, Miller 86]
I. Recursive Step

$G_0$

$G_1$
I. Recursive Step
I. Recursive Step

- Choose an arbitrary boundary node $r$
I. Recursive Step

- Choose an arbitrary boundary node $r$
- Recursively compute distances from $r$ within $G_0$
I. Recursive Step

- Choose an arbitrary boundary node $r$
- Recursively compute distances from $r$ within $G_0$
- Recursively compute distances from $r$ within $G_1$
II. Boundary-to-Boundary Distances in $G_i$

- Compute all boundary-to-boundary distances within $G_1$
  - $O(n)$ pairs of boundary nodes
  - algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-$r$ distances in $G_1$
- Repeat for $G_0$
II. Boundary-to-Boundary Distances in $G_i$

- Compute all boundary-to-boundary distances within $G_1$
  - $O(n)$ pairs of boundary nodes
  - Algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-r distances in $G_1$
- Repeat for $G_0$
II. Boundary-to-Boundary Distances in $G_i$

- Compute all boundary-to-boundary distances within $G_i$
  - $O(n)$ pairs of boundary nodes
  - algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-$r$ distances in $G_i$
- Repeat for $G_0$
II. Boundary-to-Boundary Distances in $G_i$

- Compute all boundary-to-boundary distances within $G_1$
  - $O(n)$ pairs of boundary nodes
  - Algorithm: multiple-source shortest paths [Klein 2005] in $O(n \log n)$ time
  - Uses from-$r$ distances in $G_1$
- Repeat for $G_0$
III. r-to-boundary Distances in $G$

$G_0$  $G_1$

$\delta_0$  $\delta_1$
III. r-to-boundary Distances in G

• Compute distances from r to all boundary nodes
III. r-to-boundary Distances in $G$

- Compute distances from $r$ to all boundary nodes
  - Shortest path in $G$ consists of alternating boundary-to-boundary shortest paths in $G_0$ and $G_1$
III. $r$-to-boundary Distances in $G$

- Compute distances from $r$ to all boundary nodes
  - Shortest path in $G$ consists of alternating boundary-to-boundary shortest paths in $G_0$ and $G_1$
  - "Bellman-Ford" using just boundary-to-boundary distances

$$\forall v \ e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w, v]\}$$
III. r-to-boundary Distances in G

\[ \forall v \; e_j[v] := \min_w \{ e_{j-1}[w] + \delta_i[w, v] \} \]

• All iterations in \( O \left( n^{3/2} \right) \) [Lipton-Rose-Tarjan 1979]
• \( \delta \) has a Monge non-crossing property [Fakcharoenphol-Rao 2001] \( \Rightarrow O \left( n \log^2 n \right) \) time
• We show: \( O \left( n \alpha(n) \right) \) time
III. r-to-boundary Distances in $G$

$$\forall v \ e_j[v] := \min_w \{e_{j-1}[w] + \delta_i[w, v]\}$$

- Think of a matrix whose $w, v$ element is
- We want to find all column minima of this matrix
Monge Matrices
Monge Matrices

- A matrix is Monge if for any $i \leq j, k \leq l$
  \[ \delta(i, k) + \delta(j, l) \geq \delta(i, l) + \delta(j, k) \]
Monge Matrices

• A matrix is Monge if for any $i \leq j, k \leq l$
  \[ \delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k) \]

• All column minima of an $n \times n$ Monge matrix can be found in $O(n)$ time
  [SMAWK 1989]
III. r-to-boundary Distances in $G$

$$\forall v \ e_j[v] := \min_w \{ e_{j-1}[w] + \delta_i[w, v] \}$$

- Think of a matrix whose $w,v$ element is
- We want to find all column minima of this matrix
- Show that this matrix is Monge
Crossings and the Monge Property
Crossings and the Monge Property

\[ \delta(i,k) + \delta(j,l) \]
Crossings and the Monge Property

\[ \delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k) \]
Crossings and the Monge Property

\[ \delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k) \]
Crossings and the Monge Property

\[ \delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k) \]
Crossings and the Monge Property

\[ \delta(i,k) + \delta(j,l) \geq \delta(i,l) + \delta(j,k) \]
Crossings and the Monge Property
Crossings and the Monge Property
Crossings and the Monge Property
Partial Monge Matrices

- Column Minima of a triangular Monge matrix can be found in $O(n\alpha(n))$ time [Klawe-Kleitman 1990]
III. $r$-to-boundary Distances in $G$

$\forall v \ e_j[v] := \min_w \{ e_{j-1}[w] + \delta_i[w, v] \}$

- $\delta_i$ is partially Monge even when adding $e_{j-1}[w]$ to row $w$
- Each iteration takes $O(\sqrt{n}\alpha(\sqrt{n}))$
- $O(\sqrt{n})$ iterations
- All iterations in $O(n\alpha(n))$ time
So Far We Have:

• $r$-to-boundary distances in $G$

• $r$-to-all distances in $G_i$
IV. From-r Distances in G
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths.
- Distances from r in this graph are equal to distances in G.
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths.
- Distances from r in this graph are equal to distances in G.
- Distances from r in $G_1$ are \textit{almost} feasible price function.
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths.
- Distances from r in this graph are equal to distances in G.
- Distances from r in $G_1$ are almost feasible price function.
- Setting $\phi(r)$ to a sufficiently large value makes it feasible.
IV. From-r Distances in G

- Add r-to-boundary edges. Use distances in G as edge lengths.
- Distances from r in this graph are equal to distances in G.
- Distances from r in G are almost feasible price function.
- Setting $\varphi(r)$ to a sufficiently large value makes it feasible.
## Analysis

<table>
<thead>
<tr>
<th>step</th>
<th>techniques</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.</td>
<td>recursion</td>
<td></td>
</tr>
<tr>
<td>II.</td>
<td>boundary to boundary distances in $G_i$</td>
<td>multiple-source planar shortest paths [Klein 2005]</td>
</tr>
<tr>
<td>IV.</td>
<td>distances from $r$ in $G$</td>
<td>augmented graph, feasible price function, Dijkstra</td>
</tr>
<tr>
<td>V.</td>
<td>rerooting - distances from $s$ in $G$</td>
<td>feasible price function, Dijkstra</td>
</tr>
</tbody>
</table>

$O(\log n)$ levels $\Rightarrow O(n \log^2 n)$ time

$O(n)$ space
Monge in Other Planar Problems

• Use of efficient Monge searching may be applicable in other planar graphs problem
• Example: improvement on the running time of an algorithm for the replacement path problem [Emek, Peleg, Roditty SODA08]
Thank You!