Graphs and DP

Viet Tung Hoang

The slides are loosely based on those of Dr Kevin Wayne, Princeton University and Prof. Mary Wootters, Stanford University
Agenda

1. Path Counting

2. Longest/Shortest Path in DAG

3. Parallel Scheduling

4. Shortest Paths Up to $k$ Hops
Path Counting

Given a DAG, find the number of (not necessarily shortest) paths from a source to a destination
Dynamic Programming

![Graph showing dynamic programming example with nodes 0 to 5 and arrows between them, indicating source and destination.]
Dynamic Programming

Number of paths from source to \( v \)
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

$\mathbf{x}$ Number of paths from source to $v$
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

Number of paths from source to $v$
Dynamic Programming

Number of paths from source to \( v \)
Dynamic Programming

Number of paths from source to $\nu$
Practice

Nodes are no longer indexed 0, 1, ..., so what should we do?
Arbitrary Labeling Won’t Work

Let’s say we index the nodes like this
Arbitrary Labeling Won’t Work

Can’t compute value for node 1, since the value at node 2 is not ready
How To Label?

When we process a node, we want the predecessors to be ready

Topological order
Issues In Topological Ordering

Want:
- A has index 0 (base case)
- Other nodes have indices > 0 (recursive case)

Seems impossible if there are incoming edges to source
Issues In Topological Ordering

Solution: Delete incoming edges to source

Any path coming out of the source won’t return to it (as there’s no cycle) and thus won’t contain those incoming edges
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Once Nodes Are Topologically Sorted
Agenda

1. Path Counting

2. Longest/Shortest Path in DAG

3. Parallel Scheduling

4. Shortest Paths Up to $k$ Hops
Motivation

Edges are labeled by profits

Goal: Find longest paths from the source A to maximize profits
Motivation

Edges are labeled by \textit{profits}

\textbf{Goal:} Find \textit{longest paths} from the source A to maximize profits

Longest path from A to D
What is the longest path from A to D?
Longest Path: A Pathological Case

For non-negative weights, longest paths are undefined if there are cycles.

What is the longest path from A to D?

Can make profit arbitrarily large by looping $B \rightarrow C \rightarrow E$ many times.

We’ll therefore focus on DAG.
Given a DAG with possibly negative weights, find longest paths from the source to every node.
From Longest Path To Shortest Path

Find **longest** paths from A

![Diagram showing a graph with nodes A, B, and C with edges and weights 30, 10, and 25, respectively.]

Negate weights

Find **shortest** paths from A

![Diagram showing the same graph with edges and weights negated to -30, -10, and -25, respectively.]

Unfortunately, Dijkstra **can’t** handle negative weights
Shortest-Path-DAG Algorithm

**Idea:** Use dynamic programming

**Issue:** We need an order to do recursion

How to get an order out of a DAG?
Perform a topological sort on the DAG

Want the source to have order 1:
- This should be the base case, as we already know the shortest path from source to itself
Issue In Topological Sort

Want the source to have order 1

Seems impossible if there are incoming edges to source

Solution: Delete incoming edges to source
Issue In Topological Sort

Want the source to have order 1

Seems impossible if there are incoming edges to source

**Solution:** Delete incoming edges to source

Any path coming out of the source won’t return to it (as there’s no cycle) and thus won’t contain those incoming edges.
Dynamic Programming in DAG

Topologically sorted

\[ \text{dist}[i]: \text{ distance from source to node } i \]

\[ \text{pred}[i]: \text{ predecessor in the shortest path from source to node } i \]
**Recurrence**

To reach node $i$, must go to one incoming neighbor of $i$ first

$\mathbf{d[u]} = 50$

$\mathbf{d[v]} = 45$

$\mathbf{d[i]} = \min\{50 + 7, 45 + 15\}$

$\mathbf{Y} \leftarrow \mathbf{X}$ predecessor of $X$ is $Y$

\[
\mathbf{d[i]} = \begin{cases} 
0 & \text{if } i = 1 \\
\min_{(u,i) \in E} \{d[u] + w(u, i)\} & \text{otherwise}
\end{cases}
\]
Shortest-Path-DAG via An Example
Shortest-Path-DAG via An Example
Shortest-Path-DAG via An Example

predecessor of $X$ is $Y$
Shortest-Path-DAG via An Example

predecessor of X is Y
Shortest-Path-DAG via An Example

predecessor of $X$ is $Y$
Shortest-Path-DAG via An Example

predecessor of $X$ is $Y$
Shortest-Path-DAG via An Example

predecessor of \( X \) is \( Y \)
Shortest-Path-DAG via An Example

predecessor of $X$ is $Y$
Shortest-Path-DAG via An Example

predecessor of $X$ is $Y$
Shortest-Path-DAG: Implementation

procedure Shortest-Path-DAG \((V, E, w, s)\)
Delete incoming edges to \(s\). Let \(n \leftarrow |V|\)
Topologically sort the nodes, with \(s\) of order 1.

for \(i \leftarrow 1\) to \(n\) do \(d[v] \leftarrow \infty;\) pred\([v]\) \(\leftarrow\) NULL // Initialization
\(d[1] \leftarrow 0\)

for \(i \leftarrow 2\) to \(n\) do //
    for \((u, i) \in E\) do // Look at incoming neighbors
        if \(d[i] > d[u] + w(i, u)\) then // If new dist is smaller, update
            \(d[i] \leftarrow d[u] + w(u, i);\) pred\([i]\) \(\leftarrow u\)

Time: \(O(m + n)\) \(\leftarrow\) \(n\) nodes and \(m\) edges
1. Path Counting

2. Longest/Shortest Path in DAG

3. Parallel Scheduling

4. Shortest Paths Up to $k$ Hops
Parallel Scheduling

Time to execute job A is 15

Given \( n \) jobs and unlimited number of cores, how to schedule them to minimize total latency?

Job C must finish before you can run job D
Parallel Scheduling

Total latency: 85
Parallel Scheduling

Time to execute job A is 15

Given \( n \) jobs and unlimited number of cores, how to schedule them to minimize total latency?

Job C must finish before you can run job D
Assigning Weights

Graph before transformation:
- Node A with weight 15
- Node B with weight 10
- Node C with weight 35
- Node D with weight 20
- Node E with weight 50

Graph after transformation:
- Node A with weight 15
- Node B with weight 10
- Node C with weight 35
- Node D with weight 35
- Node E with weight 35
Adding Source and Destination

Starting time

Finishing time
This path means that the running time must be at least $15 + 10 + 20 = 45$
The **longest path** from \( s \) to \( t \) is the running time for the best scheduling.
Finding Longest Paths in DAG

- The longest cost to $v$ is $x$.
- The predecessor of $X$ is $Y$. 

Diagram:

- Nodes: $s$, $A$, $B$, $C$, $D$, $E$, $t$.
- Edges with labels: $s$ to $A$ (15), $A$ to $B$ (15), $B$ to $C$ (10), $C$ to $D$ (35), $D$ to $t$ (50), $E$ to $D$ (35), $E$ to $t$ (20), $A$ to $E$ (35), $B$ to $E$ (10), $C$ to $E$ (35), $E$ to $t$ (85).
Parallel Scheduling

Total latency: 85
Agenda

1. Path Counting

2. Longest/Shortest Path in DAG

3. Parallel Scheduling

4. Shortest Paths Up to $k$ Hops
Shortest Paths Up To $k$ Hops

Give a graph without negative cycles, find the shortest path from the source to the destination using at most $k$ edges
Intuition: Replicate Graph

Dashed edges have weight 0

Only consider up to $k$ hops away
Intuition: Replicate Graphs

Shortest path in the original graph becomes a shortest path in a DAG

How to carry out this intuition (without the expensive graph transform)?
Dynamic Programming:

\[ d[i, v] : \text{ distance from A to } v \text{ using at most } i \text{ edges } \]

\[ \text{pred}[v] : \text{ predecessor of } v \text{ in the shortest path from A to } v \text{ found so far} \]
## Shortest Paths Up To $k$ Hops

![Graph with nodes A, B, C, D, E and edges with weights.

### Table: Shortest Paths

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shortest Paths Up To \( k \) Hops

![Graph Diagram]

- Source node: A
- Destinations: B, C, D, E
- Edges and weights:
  - A to B: 4
  - B to C: 3
  - C to A: -5
  - C to D: 2
  - E to D: 20
  - E to A: 8
  - D to E: -2

Table:

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shortest Paths Up To $k$ Hops

![Graph with source node A, destination nodes B, C, D, E, and paths labeled with distances.]

- **Round 0**:
  - **Source**: A, Distance: 0
  - **Destination**: B, Distance: $\infty$
  - **Destination**: C, Distance: $\infty$
  - **Destination**: D, Distance: $\infty$
  - **Destination**: E, Distance: $\infty$

- **Round 1**:
  - **Destination**: B, Distance: 0, Predecessor: A
  - **Destination**: C, Distance: 4, Predecessor: A
  - **Destination**: D, Distance: 0

- **Round 2**:
  - **Destination**: C, Distance: 4, Predecessor: B
  - **Destination**: D, Distance: 0

**Table**:

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shortest Paths Up To $k$ Hops

![Graph with nodes A, B, C, D, E and edges with weights: A to B: 4, A to C: 4, B to D: 3, C to D: 2, D to E: 20, E to C: ∞, E to D: -2, E to A: 8, B to A: -5, C to A: 0.]

predecessor of $X$ is $Y$

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Shortest Paths Up To $k$ Hops

![Graph with nodes and edges]

- Source node: A
- Destinations: B, C, D, E

**Round 0**
- A: 0
- B: $\infty$
- C: $\infty$
- D: $\infty$
- E: $\infty$

**Round 1**
- A: 0
- B: 4
- C: -5
- D: $\infty$
- E: $\infty$

**Round 2**

**Table**

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(predecessor of $X$ is $Y$)
Shortest Paths Up To $k$ Hops

![Graph with weighted edges and nodes showing shortest paths]

- **Node A** is the source node.
- **Node B** is connected to **Node A** with a weight of 4.
- **Node C** is connected to **Node A** with a weight of -5.
- **Node D** is connected to **Node A** with a weight of 2.
- **Node E** is connected to **Node A** with a weight of -2.

**Table:**

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
- The predecessor of **Node X** is **Node Y**.
- The table represents the shortest path to each destination node from the source node at each round.
Shortest Paths Up To $k$ Hops

\begin{array}{c|c|c|c|c|c}
\text{Round} & \text{Dest} & A & B & C & D & E \\
\hline
0 & 0 & \infty & \infty & \infty & \infty & \infty \\
1 & 0 & 4 & -5 & \infty & \infty & \infty \\
2 & 0 & 4 & & & & \\
\end{array}

The predecessor of $X$ is $Y$. 

Source Node: A

Nodes: A, B, C, D, E

Weights:
- A to B: 4
- A to C: 4
- A to D: \infty
- A to E: \infty
- B to C: 3
- B to D: 20
- B to E: \infty
- C to D: 2
- C to E: \infty
- D to E: -2

Dijkstra's Algorithm Applied:

Round 0:
- From A to itself: 0
- From A to B: 4
- From A to C: \infty
- From A to D: \infty
- From A to E: \infty

Round 1:
- From B to C: 3
- From B to D: 20
- From B to E: \infty

Round 2:
- From B to E: \infty

The table represents the shortest paths up to $k$ hops for each node.
Shortest Paths Up To $k$ Hops

**Diagram:**
- **Source:** A
- **Nodes:** B, C, D, E
- **Weights:**
  - AB: 4
  - AC: -5
  - BC: 3
  - BD: 20
  - BE: ∞
  - CD: 2
  - CE: 8
  - DE: ∞

**Table:**

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
- Predecessor of $X$ is $Y$
Shortest Paths Up To $k$ Hops

![Graph showing shortest paths up to $k$ hops]

**Table:**

<table>
<thead>
<tr>
<th>Round</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>-3</td>
<td></td>
</tr>
</tbody>
</table>
Shortest Paths Up To $k$ Hops

The table below summarizes the shortest paths from the source to each destination up to $k$ hops.

<table>
<thead>
<tr>
<th>Round</th>
<th>Dest</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
<td>4</td>
<td>-5</td>
<td>-3</td>
<td>12</td>
</tr>
</tbody>
</table>
Shortest Paths Up To $k$ Hops

Using up to 2 hops, this is the shortest path from A to E