Decremental All Pairs ALL Shortest Paths

Meghana Nasre.
Department of Computer Science and Engineering,
IIT Madras.

joint work with
Matteo Pontecorvi and Vijaya Ramachandran

CALDAM 2015
Pre-Conference School on Discrete Mathematics
Feb 5, 2015.
All Pairs Shortest Paths Problem

**Input:** Directed graph with positive edge weights.

**Goal:** Compute APSP distances.

- Directed graph with positive edge weights.
- **ALL Pairs Shortest Paths (APSP) Algorithm**
- **Distance Matrix**
Dynamic APSP

**Input:** Directed graph with positive edge weights.
Dynamic APSP

**Input:** Directed graph with positive edge weights.

- Change in the graph.
- **Goal:** Maintain distance matrix efficiently.
Dynamic APSP

**Input**: Directed graph with positive edge weights.

- Change in the graph.
- **Goal**: Maintain distance matrix efficiently.
  - Update time.
  - Query time.
Dynamic APSP

Simple Approaches

1. Do nothing
   - Update time: $O(1)$.
   - Query time: $O(n^2)$.

2. Do everything
   - Update time: $O(n^3)$.
   - Query time: $O(1)$.
Decremental All Pairs ALL Shortest Paths

Introduction

Dynamic APSP

Simple Approaches

1. Do nothing
   - Update time: $O(1)$.
   - Query time: $O(n^2)$.

2. Do everything
   - Update time: $O(n^3)$.
   - Query time: $O(1)$.

Can we trade off between update time and query time?
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1 Incremental.
   - Edge additions/ edge weight decreases.
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1 Incremental.
- Edge additions/ edge weight decreases.

2 Decremental.
- Edge deletions/ edge weight increases.
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1 Incremental.
   - Edge additions/ edge weight decreases.

2 Decremental.
   - Edge deletions/ edge weight increases.

3 Fully Dynamic.
   - Interleaved sequence of incremental and decremental updates.
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

**Incremental:**
- Edge additions/ edge weight decreases.
- Straightforward: \( O(n^2) \) update time, \( O(1) \) query time.
- All earlier paths continue to exist!
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1 Incremental:
   - Edge additions/ edge weight decreases.
   - straightforward: \( O(n^2) \) update time, \( O(1) \) query time.
   - all earlier paths continue to exist!

2 Decremental/Fully-Dynamic : several challenges!
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1. **Incremental:**
   - Edge additions/ edge weight decreases.
   - Straightforward: \( O(n^2) \) update time, \( O(1) \) query time.
   - All earlier paths continue to exist!

2. **Decremental/Fully-Dynamic**: several challenges!
   - Even Shiloach Trees (unweighted BFS trees).
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1 Incremental:
   - Edge additions/ edge weight decreases.
   - Straightforward: \( O(n^2) \) update time, \( O(1) \) query time.
   - All earlier paths continue to exist!

2 Decremental/Fully-Dynamic: several challenges!
   - Even Shiloach Trees (unweighted BFS trees).
   - Restricted graph classes (planar, max-weight bounded, \ldots)
Dynamic Updates

A sequence of updates

\[ u_1, u_2, \ldots, u_k \]

1 Incremental:
   - Edge additions/ edge weight decreases.
   - straightforward: \( O(n^2) \) update time, \( O(1) \) query time.
   - all earlier paths continue to exist!

2 Decremental/Fully-Dynamic : several challenges!
   - Even Shiloach Trees (unweighted BFS trees).
   - Restricted graph classes (planar, max-weight bounded, \ldots)
   - \( O(n^2 \cdot \text{polylog}(n)) \) update time, \( O(1) \) query time.
   - Demetrescu and Italiano (STOC 2003, JACM 2004).
Outline of the talk

1. Decremental APSP.
   - Locally shortest paths.
   - Properties of LSPs.
   - An update algorithm using LSP.

2. Decremental AP-ALL-SP.
   - Maintain all shortest paths, a count matrix.
   - Adapt decremental APSP to APASP.

3. Application to Betweenness Centrality.
Decremental APSP
Decremental updates

**Input:** Directed graph $G = (V, E)$ with positive edge weights.

**Goal:** Maintain distance matrix.

![Diagram of a graph with vertices and edges labeled with weights]
Some intuition

Input: Directed graph $G = (V, E)$ with positive edge weights.
Goal: Maintain distance matrix.
Some intuition

**Input:** Directed graph $G = (V, E)$ with positive edge weights.

**Goal:** Maintain distance matrix.

What distances have changed?

- $x \ldots v$
Some intuition

**Input:** Directed graph $G = (V, E)$ with positive edge weights.

**Goal:** Maintain distance matrix.

What distances have changed?

- $x' \ldots v$
- $x' \ldots x$
Some intuition

**Input:** Directed graph $G = (V, E)$ with positive edge weights.

**Goal:** Maintain distance matrix.

What distances have changed?

- $x \ldots v$
- $x' \ldots x$
- $x \ldots a_2$
Some intuition

**Input:** Directed graph $G = (V, E)$ with positive edge weights.

**Goal:** Maintain distance matrix.

What distances have changed?

- $x \ldots v$
- $x' \ldots x$
- $x \ldots a_2$

Decremental updates: $\Rightarrow$

Distances can increase or remain same; cannot decrease!!!
Some intuition

Input: Directed graph \( G = (V, E) \) with positive edge weights.
Goal: Maintain distance matrix.

What distances have changed?

- \( x \ldots v \)
- \( x' \ldots x \)
- \( x \ldots a_2 \)

Decremental updates: \( \Rightarrow \)
distances can increase or remain same; cannot decrease!!!

Does it help to have next shortest paths and so on?
Some intuition

**Input:** Directed graph $G = (V, E)$ with positive edge weights.

**Goal:** Maintain distance matrix.

What distances have changed?

- $x \ldots v$
- $x' \ldots x$
- $x \ldots a_2$

Decremental updates: $\Rightarrow$
- Distances can increase or remain same; cannot decrease!!!

- Does it help to have next shortest paths and so on?
- **Pitfall:** Next has to be without knowledge of updates!
Locally Shortest Paths

Demetrescu and Italiano (JACM 2004)
- introduced an elegant idea of locally shortest paths (LSPs).
Decremental All Pairs ALL Shortest Paths

Decremental APSP

Locally Shortest Paths

Demetrescu and Italiano (JACM 2004)

- introduced an elegant idea of locally shortest paths (LSPs).

When is a path $p$ an LSP?

- $p$ is a single vertex/edge, OR
- every proper sub-path of $p$ is a shortest path.
Locally Shortest Paths

Demetrescu and Italiano (JACM 2004)
- introduced an elegant idea of locally shortest paths (LSPs).

When is a path $p$ an LSP?

- $p$ is a single vertex/ edge, OR
- every proper sub-path of $p$ is a shortest path.

$SP \subseteq LSP$
**LSP: example**

All edges are trivially LSPs.

\[ x \rightarrow a_1 \rightarrow a_2 \text{ is an LSP.} \]

\[ x' \rightarrow x \rightarrow a_1 \rightarrow a_2 \text{ is NOT an LSP.} \]

Between a pair of vertices there can be multiple LSPs of different weights. The min-weight LSPs are SPs.

LSPs are defined independent of the update.
LSP: example

- All edges are trivially LSPs.
LSP: example

- All edges are trivially LSPs.
- $x \to a_1 \to a_2$ is an LSP.
LSP: example

- All edges are trivially LSPs.
- $x \rightarrow a_1 \rightarrow a_2$ is an LSP.
- $x' \rightarrow x \rightarrow a_1 \rightarrow a_2$ is NOT an LSP.
LSP: example

- All edges are trivially LSPs.
- $x \rightarrow a_1 \rightarrow a_2$ is an LSP.
- $x' \rightarrow x \rightarrow a_1 \rightarrow a_2$ is NOT an LSP.
- Between a pair of vertices there can be multiple LSPs of different weights.
- The min-weight LSPs are SPs.
LSP: example

- All edges are trivially LSPs.
- $x \rightarrow a_1 \rightarrow a_2$ is an LSP.
- $x' \rightarrow x \rightarrow a_1 \rightarrow a_2$ is NOT an LSP.
- Between a pair of vertices there can be multiple LSPs of different weights.
- The min-weight LSPs are SPs.
- LSPs are defined independent of the update.
Why are LSPs useful?

Shortest path from $x$ to $a_2$ before update: $x \rightarrow a_2$.

After update: $x \rightarrow a_1 \rightarrow a_2$.

An LSP before update $A$ nice to have statement: $G$: before update; $G'$: after update.

A path $p$ is a shortest path in $G'$ if $p$ is an LSP in $G$. 

14 / 27
Why are LSPs useful?

Shortest path from $x$ to $a_2$

- before update: $x \rightarrow a_2$.
- after update: $x \rightarrow a_1 \rightarrow a_2$.

an LSP before update
Why are LSPs useful?

A nice to have statement:

- \( G \): before update; \( G' \): after update.
- A path \( p \) is a shortest path in \( G' \) if \( p \) is an LSP in \( G \).
Why are LSPs useful?

A nice to have statement:

- \( G \): before update; \( G' \): after update.
- A path \( p \) is a shortest path in \( G' \) if \( p \) is an LSP in \( G \).

Consider \( p = x' \rightarrow x \rightarrow a_1 \rightarrow a_2 \).
Why are LSPs useful?

**Theorem [DI 2004]:**

- $G$: before update; $G'$: after update.
- A path $p$ is a shortest path in $G'$ if $p$ has a sub-path $p'$ which is an LSP in $G$. 
Why are LSPs useful?

**Theorem [DI 2004]:**

- $G$: before update; $G'$: after update.
- A path $p$ is a shortest path in $G'$ if $p$ has a sub-path $p'$ which is an LSP in $G$.

Suggests the following mechanism:

- Maintain LSPs under updates.
- Allow “extending” LSPs to create longer paths.
Decremental All Pairs ALL Shortest Paths

Data structures
Data structures

- for every $x, y$: $P_{xy}, P^*_{xy}$.
  - $P_{xa_2} = \{x \rightarrow a_2, x \rightarrow a_1 \rightarrow a_2\}$.
  - $P^*_{xa_2} = \{x \rightarrow a_2\}$.
Data structures

- for every $x, y$: $P_{xy}, P^*_{xy}$.
  - $P_{xa_2} = \{x \rightarrow a_2, x \rightarrow a_1 \rightarrow a_2\}$.
  - $P^*_{xa_2} = \{x \rightarrow a_2\}$.
- Ability to “extend” paths.
for every $x, y$: $P_{xy}, P^*_{xy}$.
- $P_{xa_2} = \{x \rightarrow a_2, x \rightarrow a_1 \rightarrow a_2\}$.
- $P^*_{xa_2} = \{x \rightarrow a_2\}$.

Ability to "extend" paths.
for every LSP $p$: $L(p), L^*(p)$.
- $L(a_1 \rightarrow a_2) = \{x\}$.
- $L^*(a_1 \rightarrow a_2) = \emptyset$. 
Data structures

- for every $x, y$: $P_{xy}, P^*_{xy}$.
  - $P_{xa_2} = \{x \rightarrow a_2, x \rightarrow a_1 \rightarrow a_2\}$.
  - $P^*_{xa_2} = \{x \rightarrow a_2\}$.

- Ability to “extend” paths.
- for every LSP $p$: $L(p), L^*(p)$.
  - $L(a_1 \rightarrow a_2) = \{x\}$.
  - $L^*(a_1 \rightarrow a_2) = \emptyset$.

- Similarly define $R(p), R^*(p)$.
Data structures

- for every $x, y$: $P_{xy}, P^*_{xy}$.
  - $P_{xa_2} = \{x \rightarrow a_2, x \rightarrow a_1 \rightarrow a_2\}$.
  - $P^*_{xa_2} = \{x \rightarrow a_2\}$.

- Ability to “extend” paths.

- for every LSP $p$: $L(p), L^*(p)$.
  - $L(a_1 \rightarrow a_2) = \{x\}$.
  - $L^*(a_1 \rightarrow a_2) = \emptyset$.

- Similarly define $R(p), R^*(p)$.

**Goal:** Maintain these under decremental updates.
Update algorithm – sketch

- Data Structures:
  - for every $x, y$: $P_{xy}, P^*_{xy}$
  - for every LSP $p$: $L(p), L^*(p), R(p), R^*(p)$.
Update algorithm – sketch

- Data Structures:
  - for every $x, y$: $P_{xy}, P_{xy}^*$
  - for every LSP $p$: $L(p), L^*(p), R(p), R^*(p)$.

- Decremental update on $v$
Update algorithm – sketch

Data Structures:
- for every $x, y$: $P_{xy}, P^*_{xy}$
- for every LSP $p$: $L(p), L^*(p), R(p), R^*(p)$.

Decremental update on $v$

Update algorithm
1. Cleanup
   - removes all LSPs that contain $v$.
   - deals with only those LSPs that contain $v$. 
Decremental All Pairs ALL Shortest Paths

Decremental APSP

Update algorithm – sketch

- **Data Structures:**
  - for every $x, y$: $P_{xy}, P_{xy}^*$
  - for every LSP $p$: $L(p), L^*(p), R(p), R^*(p)$.

- **Decremental update on $v$**

- **Update algorithm**
  1. **Cleanup**
     - removes all LSPs that contain $v$.
     - deals with *only* those LSPs that contain $v$.
  2. **Fixup**
     - adds back LSPs (start with single edges) that “become” LSPs.
     - “extends” new shortest paths to form “new” LSPs.
Update algorithm – sketch

- **Data Structures:**
  - for every \(x, y\): \(P_{xy}, P^*_{xy}\)
  - for every LSP \(p\): \(L(p), L^*(p), R(p), R^*(p)\).

- **Decremental update on \(v\)**

- **Update algorithm**
  1. **Cleanup**
     - removes all LSPs that contain \(v\).
     - deals with only those LSPs that contain \(v\).
  2. **Fixup**
     - adds back LSPs (start with single edges) that “become” LSPs.
     - “extends” new shortest paths to form “new” LSPs.

- **Why should this work? Aren’t there too many LSP??**
Counting LSPs

- Assume unique shortest paths between every pair. \textit{w.l.o.g.}
Counting LSPs

- Assume unique shortest paths between every pair. w.l.o.g.
- An LSP is uniquely defined by first edge (xa) and end vertex y.
Counting LSPs

- Assume unique shortest paths between every pair. \( \text{w.l.o.g.} \)
- An LSP is uniquely defined by first edge \( (xa) \) and end vertex \( y \).
- How many LSPs are there?
  - \# of LSPs = \( O(mn) \).
Counting LSPs

- Assume unique shortest paths between every pair. w.l.o.g.
- An LSP is uniquely defined by first edge \((xa)\) and end vertex \(y\).
- How many LSPs are there?
  - \# of LSPs = \(O(mn)\).
- How many LSPs do we consider during our algorithm?
Decremental All Pairs ALL Shortest Paths

Decremental APSP

Counting LSPs

- Assume unique shortest paths between every pair. \( \text{w.l.o.g.} \)
- An LSP is uniquely defined by first edge \((xa)\) and end vertex \(y\).
- How many LSPs are there?
  - \# of LSPs = \(O(mn)\).
- How many LSPs do we consider during our algorithm?
- During cleanup: only those that contain \(v\).
  - LSPs that start and end at \(v\): \(O(n^2)\).
  - LSPs that contain \(v\) as intermediate vertex: \(O(n^2)\) again.
Counting LSPs

- Assume unique shortest paths between every pair. \( \text{w.l.o.g.} \)
- An LSP is uniquely defined by first edge \((xa)\) and end vertex \(y\).
- How many LSPs are there?
  - \(\# \text{ of LSPs} = O(mn)\).
- How many LSPs do we consider during our algorithm?
  - During cleanup: only those that contain \(v\).
    - LSPs that start and end at \(v\): \(O(n^2)\).
    - LSPs that contain \(v\) as intermediate vertex: \(O(n^2)\) again.
  - During fixup: possibly more – but can be bounded over a sequence. \(O(n^2)\) amortized over \(\Omega(m/n)\) updates.
Counting LSPs

- Assume unique shortest paths between every pair. \textit{w.l.o.g.}
- An LSP is uniquely defined by \textit{first edge} \((xa)\) and \textit{end vertex} \(y\).
- How many LSPs are there?
  - \# of LSPs = \(O(mn)\).
- How many LSPs do we consider during our algorithm?
  - During cleanup: only those that contain \(v\).
    - LSPs that start and end at \(v\): \(O(n^2)\).
    - LSPs that contain \(v\) as intermediate vertex: \(O(n^2)\) again.
  - During fixup: possibly more – but can be bounded over a sequence. \(O(n^2)\) amortized over \(\Omega(m/n)\) updates.

\textbf{Unique shortest path} assumption \textit{crucial} to get bounds.
Counting LSPs

- Assume unique shortest paths between every pair. \( \text{w.l.o.g.} \)
- An LSP is uniquely defined by first edge \((xa)\) and end vertex \(y\).
- How many LSPs are there?
  - \# of LSPs = \(O(mn)\).
- How many LSPs do we consider during our algorithm?
  - During cleanup: only those that contain \(v\).
    - LSPs that start and end at \(v\): \(O(n^2)\).
    - LSPs that contain \(v\) as intermediate vertex: \(O(n^2)\) again.
  - During fixup: possibly more – but can be bounded over a sequence. \(O(n^2)\) amortized over \(\Omega(m/n)\) updates.

Unique shortest path assumption crucial to get bounds.

Can we get over it?
Decremental APASP
Decremental APASP

**Input:** Directed graph with positive edge weights.

- **Goal 1:** Maintain distance matrix efficiently.
- **Goal 2:** Maintain count matrix efficiently.
Decremental APASP

- Generalize the DI method \textit{without unique shortest paths} assumption.
- Define a succinct representation of LSPs – \textit{locally shortest tuples (LST)}. 

Tuple: Set of paths with same first and last edge.
Decremental APASP

- Generalize the DI method \textit{without unique shortest paths} assumption.
- Define a succinct representation of LSPs – \textit{locally shortest tuples} (LST).

\textbf{Tuple:} Set of paths with same first and last edge.
Decremental APASP

- Generalize the DI method without unique shortest paths assumption.
- Define a succinct representation of LSPs – locally shortest tuples (LST).

Tuple: Set of paths with same first and last edge.

- A decremental update algorithm which works on LSTs.
Decremental APASP

- Generalize the DI method without unique shortest paths assumption.
- Define a succinct representation of LSPs – locally shortest tuples (LST).

Tuple: Set of paths with same first and last edge.

- A decremental update algorithm which works on LSTs.
- How many tuples are there?
Counting LSTs

Definitions:

- for $v$, $E_v^*$: edges that lie on shortest paths through $v$.
- for $v$, $I_v^*$: edges that are incoming to $v$. 
Counting LSTs

Definitions:

- for $v$, $E^*_v$: edges that lie on shortest paths through $v$.
- for $v$, $I^*_v$: edges that are incoming to $v$.
- $\nu^*$ : $\max_{v \in V} \{|E^*_v|\}$. 
Counting LSTs

Definitions:

- for $v$, $E_v^*$: edges that lie on shortest paths through $v$.
- for $v$, $I_v^*$: edges that are incoming to $v$.
- $\nu^*$: $\max_{v \in V} \{|E_v^*|\}$.
  - $\nu^*$ is $O(n)$ when every pair has constant number of SPs.
Counting LSTs

Definitions:
- for $v$, $E_v^*$: edges that lie on shortest paths through $v$.
- for $v$, $I_v^*$: edges that are incoming to $v$.
- $\nu^* : \max_{v \in V} \{|E_v^*|\}$.
  - $\nu^*$ is $O(n)$ when every pair has constant number of SPs.
- $E^* = \bigcup_{v \in V} E_v^*$.
- $m^* = |E^*|$.
Counting LSTs

Definitions:

- for $v$, $E^*_v$: edges that lie on shortest paths through $v$.
- for $v$, $I^*_v$: edges that are incoming to $v$.
- $\nu^* : \max_{v \in V} \{|E^*_v|\}$.
  - $\nu^*$ is $O(n)$ when every pair has constant number of SPs.
- $E^* = \bigcup_{v \in V} E^*_v$.
- $m^* = |E^*|$.
  - $m^*$ is always upper bounded by $m$. 
Counting LSTs

Definitions:

- for $v$, $E_v^*$: edges that lie on shortest paths through $v$.
- for $v$, $I_v^*$: edges that are incoming to $v$.
- $\nu^* : \max_{v \in V} \{|E_v^*|\}$.
  - $\nu^*$ is $O(n)$ when every pair has constant number of SPs.
- $E^* = \bigcup_{v \in V} E_v^*$.
- $m^* = |E^*|$.  
  - $m^*$ is always upper bounded by $m$. 
Decremental All Pairs ALL Shortest Paths

Counting LSTs

- $\nu^* : \max_{v \in V} \{|E_v^*|\}$. 
- $m^* = |E^*|$. 

How many LSTs?

- LSTs of form $(\times a, \times \times)$. 

Decremental All Pairs All Shortest Paths

Counting LSTs

- \( \nu^* : \max_{v \in V} \{|E_v^*|\} \).
- \( m^* = |E^*| \).

How many LSTs?

- LSTs of form \((\times a, \times \times)\).
- Last edge can be chosen in \(\nu^*\) ways.
Counting LSTs

- \( \nu^* : \max_{v \in V} \{|E^*_v|\} \).
- \( m^* = |E^*| \).

How many LSTs?

- LSTs of form \((\times a, \times \times)\).
- Last edge can be chosen in \( \nu^* \) ways.
- First edge: lies on SP through \( a \) and is incoming to \( a \).
  - First edge can be chosen in: \( |E^*_a \cap I_a| \) ways.
Counting LSTs

- $\nu^* : \max_{v \in V} \{|E_v^*|\}$.
- $m^* = |E^*|$.

How many LSTs?

- LSTs of form $(\times a, \times \times)$.
- Last edge can be chosen in $\nu^*$ ways.
- First edge: lies on SP through $a$ and is incoming to $a$.
  - First edge can be chosen in: $|E_a^* \cap I_a|$ ways.
- # of LSTs: $O(m^* \cdot \nu^*)$. 

Maintain LSTs instead of LSPs using similar algorithm.
Since we maintain ALL paths, counts matrix is maintained.
Decremental All Pairs ALL Shortest Paths

Decremental APASP

Counting LSTs

- \( \nu^* : \max_{v \in V} \{|E_v^*|\} \).
- \( m^* = |E^*| \).

How many LSTs?

- LSTs of form \((\times a, \times \times)\).
- Last edge can be chosen in \( \nu^* \) ways.
- First edge: lies on SP through \( a \) and is incoming to \( a \).
  - First edge can be chosen in: \(|E_a^* \cap I_a|\) ways.
- \# of LSTs: \( O(m^* \cdot \nu^*) \).
- \# of LSTs that contain \( v \): \( O((\nu^*)^2) \).
Counting LSTs

- $\nu^* : \max_{v \in V} \{|E^*_v|\}$.
- $m^* = |E^*|$.

How many LSTs?

- LSTs of form ($\times a$, $\times \times$).
- Last edge can be chosen in $\nu^*$ ways.
- First edge: lies on SP through $a$ and is incoming to $a$.
  - First edge can be chosen in: $|E^*_a \cap l_a|$ ways.
- # of LSTs: $O(m^* \cdot \nu^*)$.
- # of LSTs that contain $v$: $O((\nu^*)^2)$.

Maintain LSTs instead of LSPs using similar algorithm. Since we maintain ALL paths, counts matrix is maintained.
What is Betweenness Centrality?

**Input:** A directed graph $G = (V, E)$; positive real edge weights.

- $\sigma_{st}$ : # SPs from $s$ to $t$.
- $\sigma_{st}(v)$ : # SPs from $s$ to $t$ that pass via $v$. 

Decremental All Pairs ALL Shortest Paths

Betweenness Centrality

Decremental All Pairs ALL Shortest Paths

Betweenness Centrality
What is Betweenness Centrality?

**Input:** A directed graph $G = (V, E)$; positive real edge weights.

- $\sigma_{st}$ : # SPs from $s$ to $t$.
- $\sigma_{st}(v)$ : # SPs from $s$ to $t$ that pass via $v$.

$$BC(v) = \sum_{s,t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$
What is Betweenness Centrality?

**Input:** A directed graph $G = (V, E)$; positive real edge weights.

- $\sigma_{st}$ : # SPs from $s$ to $t$.
- $\sigma_{st}(v)$ : # SPs from $s$ to $t$ that pass via $v$.

$$BC(v) = \Sigma_{s,t\neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$

- A useful measure in analysis of large networks.

**Computing BC for all vertices.**

- Brandes’ Algorithm (2001), variant of Dijkstra’s SSSP.
What is Betweenness Centrality?

**Input:** A directed graph $G = (V, E)$; positive real edge weights.

- $\sigma_{st}$: \# SPs from $s$ to $t$.
- $\sigma_{st}(v)$: \# SPs from $s$ to $t$ that pass via $v$.

$$BC(v) = \sum_{s,t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}.$$ 

- A useful measure in analysis of large networks.

**Computing BC for all vertices.**

- Brandes’ Algorithm (2001), variant of Dijkstra’s SSSP.
  - $n$ executions of Dijkstra’s: $O(mn + n^2 \log n)$ time.
- **Decremental BC:** using decremental APASP.
Summary

- Dynamic APSP update algorithm.
Summary

- Dynamic APSP update algorithm.
- Locally shortest paths.
Summary

- Dynamic APSP update algorithm.
- Locally shortest paths.
- Storing a super-set of shortest paths does the trick!
Summary

- Dynamic APSP update algorithm.
- Locally shortest paths.
- Storing a super-set of shortest paths does the trick!
- ALL shortest paths can be maintained (locally shortest path tuples).
Summary

- Dynamic APSP update algorithm.
- Locally shortest paths.
- Storing a super-set of shortest paths does the trick!
- ALL shortest paths can be maintained (locally shortest path tuples).
- Space issues need to be addressed.
Summary

- Dynamic APSP update algorithm.
- Locally shortest paths.
- Storing a super-set of shortest paths does the trick!
- ALL shortest paths can be maintained (locally shortest path tuples).
- Space issues need to be addressed.
- Connection to other shortest paths related problems.
Thank You!!