Efficient Algorithms for Path Problems in Weighted Graphs

Virginia Vassilevska

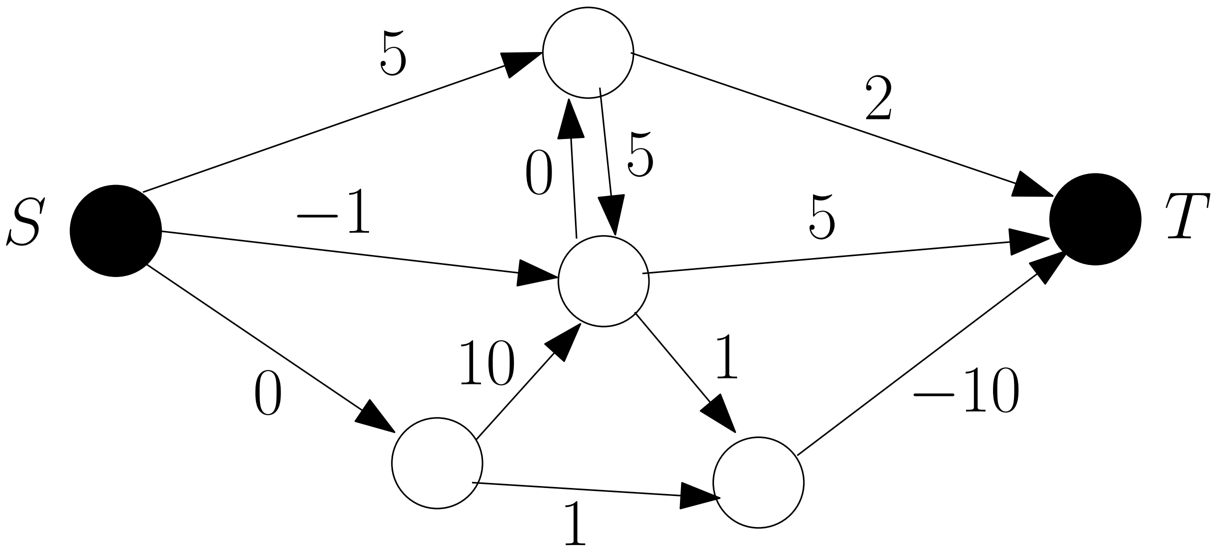
Thesis Proposal

November 14, 2007

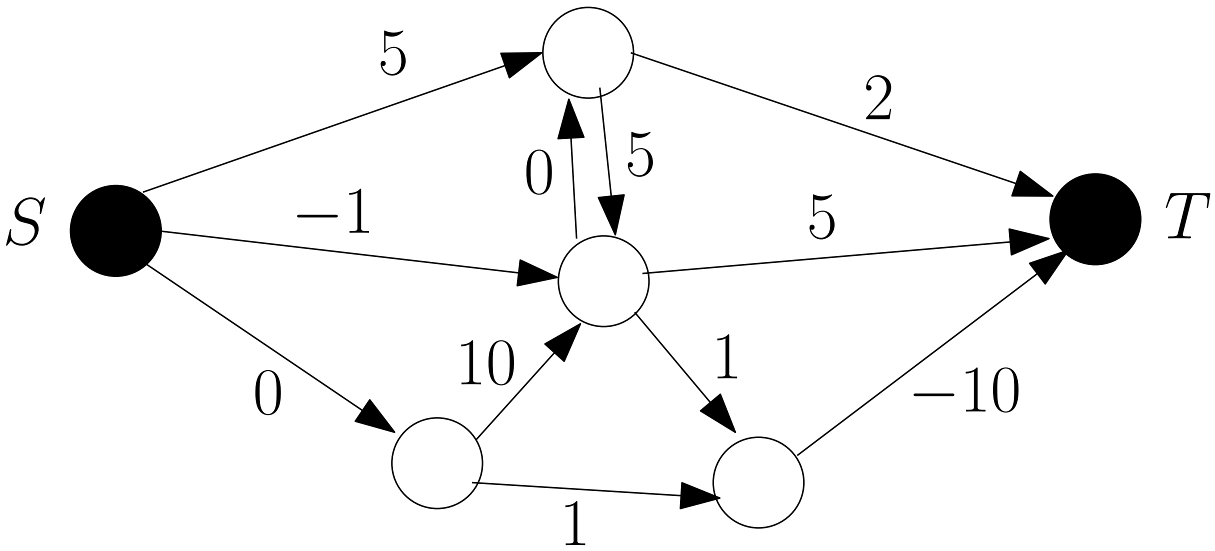
Thesis Committee:

Guy Blelloch, Manuel Blum, Anupam Gupta, Uri Zwick (Tel Aviv University)

Weighted Graph Path Problems – Introduction



Weighted Graph Path Problems – Introduction

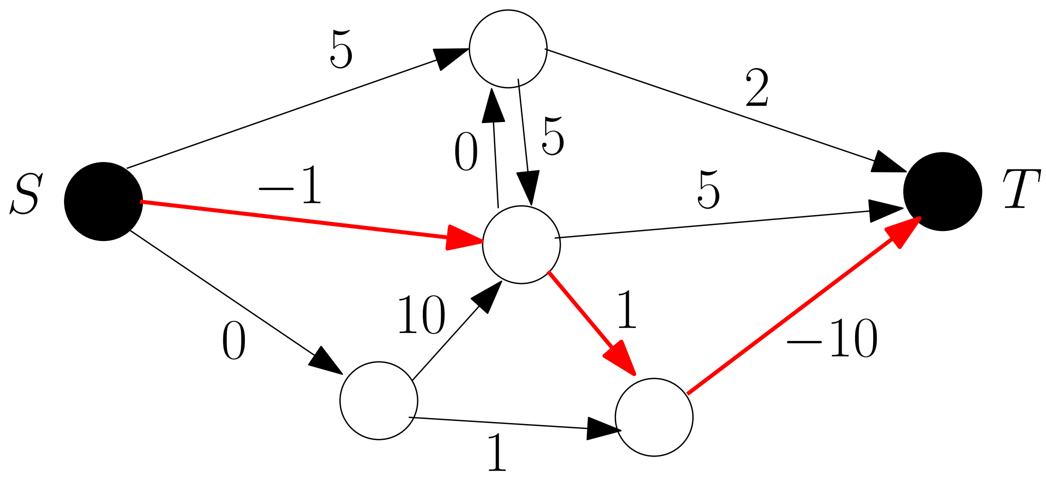


Our Problem: find a path $S = v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k = T$ optimizing a given measure.

Path Measures

Shortest Paths: Find $S = v_0 \to v_1 \to v_2 \to \ldots \to v_k = T$ minimizing

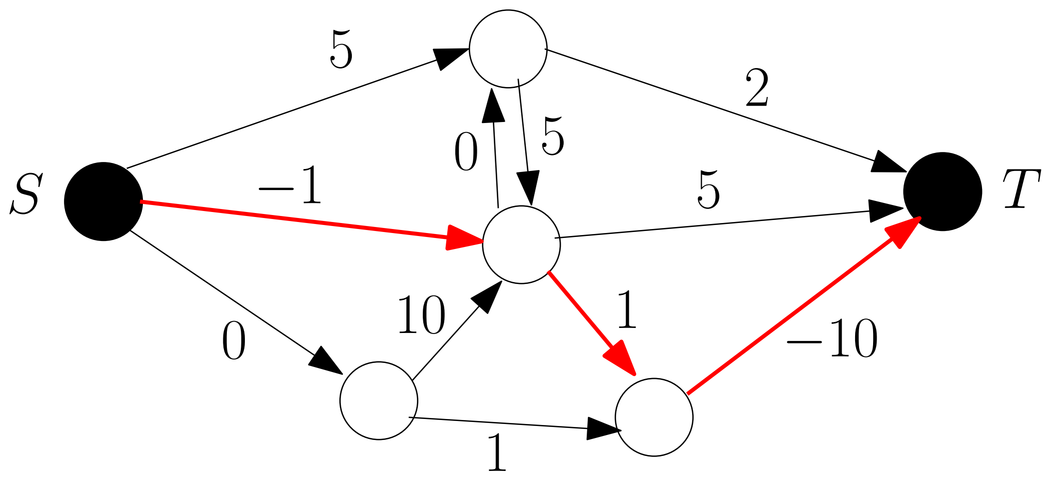
$$\sum_{i=0}^{k-1} w(v_i, v_{i+1}).$$



Path Measures

Shortest Paths: Find $S = v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k = T$ minimizing

$$\sum_{i=0}^{k-1} w(v_i, v_{i+1}).$$



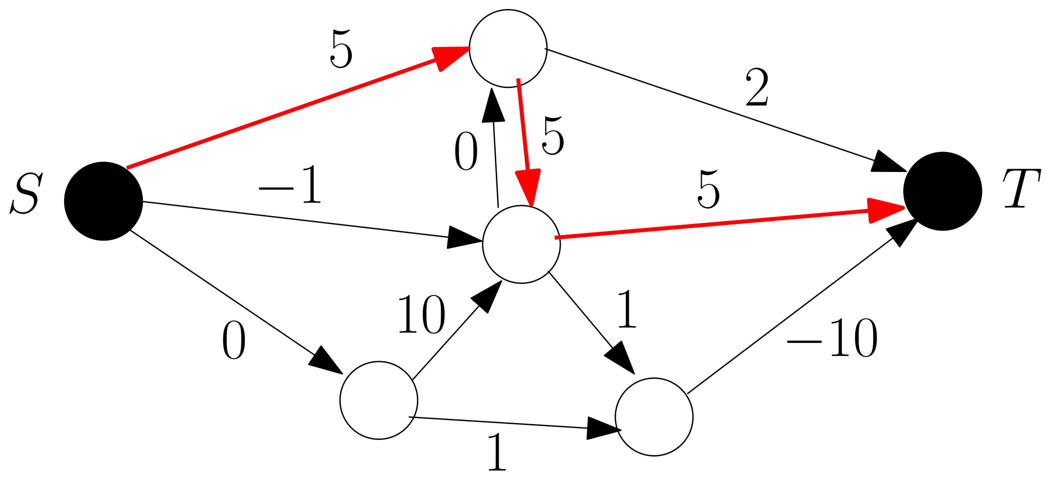
Application: find shortest road distance between two cities on a map.

Path Measures Cont.

Maximum Bottleneck Paths:

Find $S = v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k = T$ maximizing

$$\min_{i=0}^{k-1} w(v_i, v_{i+1}).$$

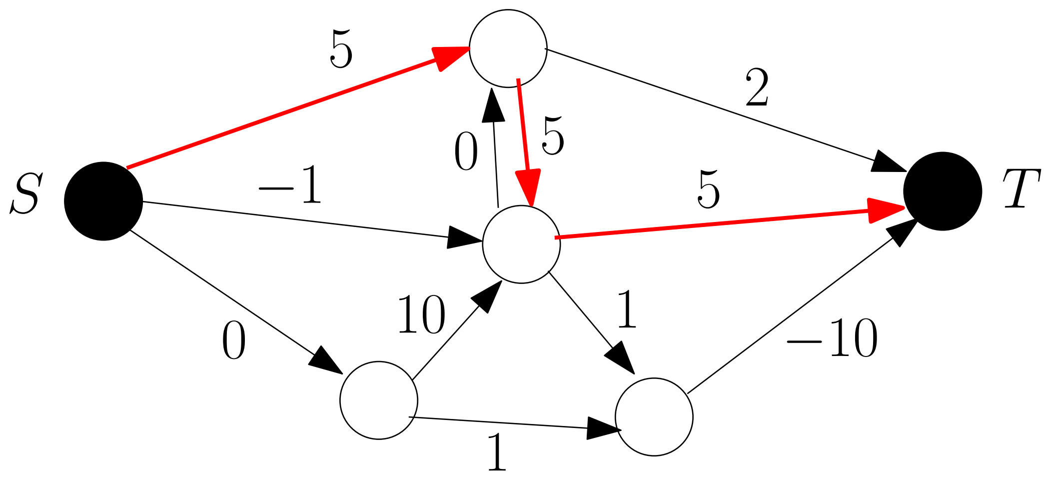


Path Measures Cont.

Maximum Bottleneck Paths:

Find $S = v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k = T$ maximizing

$$\min_{i=0}^{k-1} w(v_i, v_{i+1}).$$

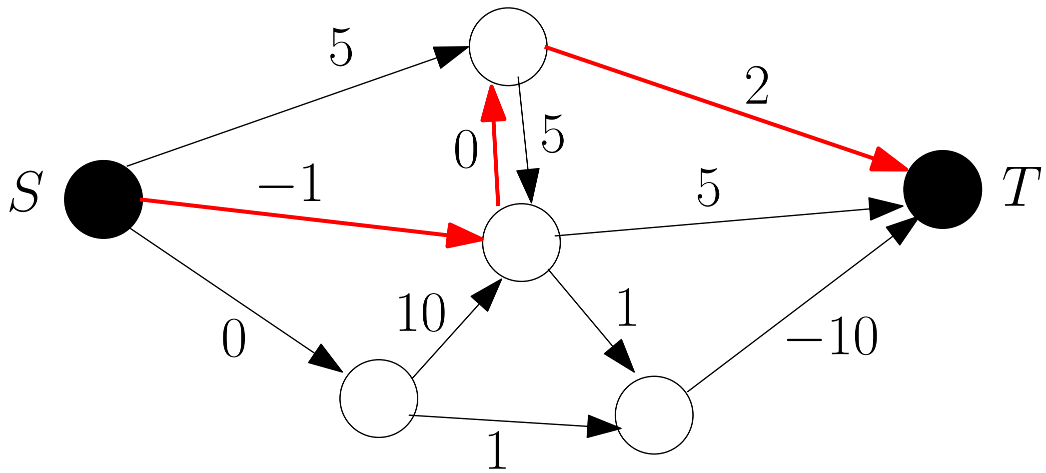


Application: find road path of highest tunnel clearance between two cities.

Path Measures Cont.

Minimum Nondecreasing Paths:

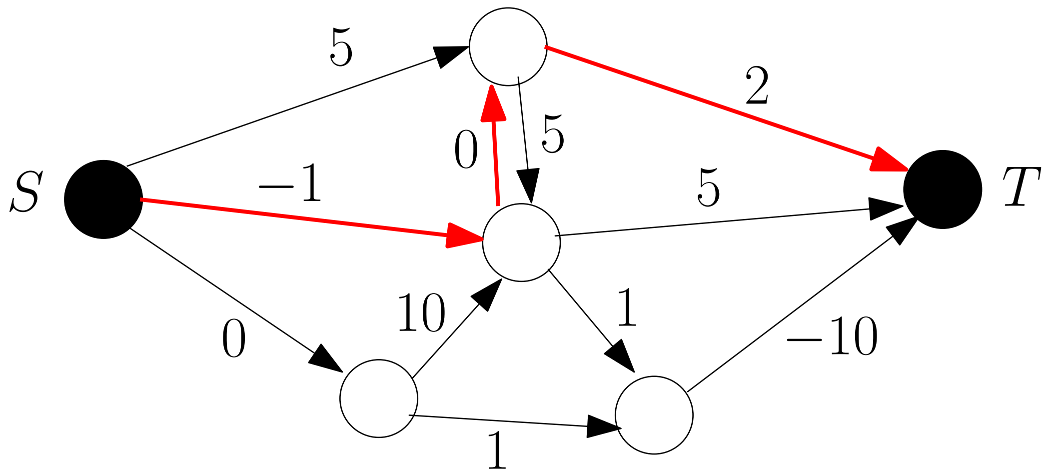
Find $S = v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_{k-1} \longrightarrow v_k = T$ such that $w(v_i, v_{i+1}) \leq w(v_{i+1}, v_{i+2})$ for all i, minimizing $w(v_{k-1}, T)$.



Path Measures Cont.

Minimum Nondecreasing Paths:

Find $S = v_0 \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_{k-1} \longrightarrow v_k = T$ such that $w(v_i, v_{i+1}) \leq w(v_{i+1}, v_{i+2})$ for all i, minimizing $w(v_{k-1}, T)$.

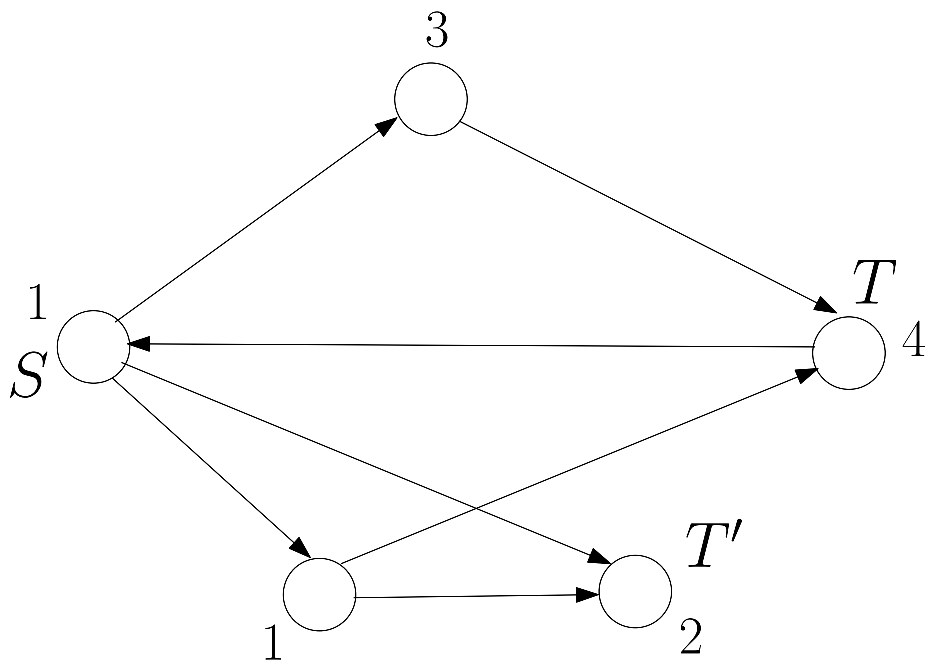


Application: compute train itinerary which gets you from one city to another as early as possible.

Path Measures Cont.

Maximum Node Weighted Triangle:

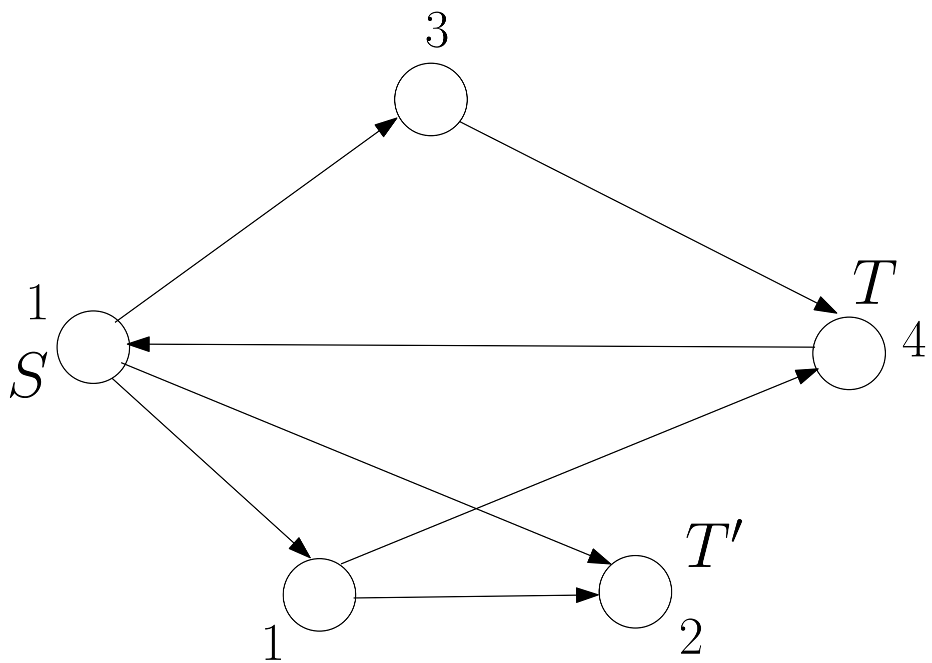
In a node-weighted graph, if (T, S) is an edge, find $S \rightarrow v \rightarrow T$ maximizing $w(S) + w(v) + w(T)$.



Path Measures Cont.

Maximum Node Weighted Triangle:

In a node-weighted graph, if (T, S) is an edge, find $S \to v \to T$ maximizing $w(S) + w(v) + w(T)$.

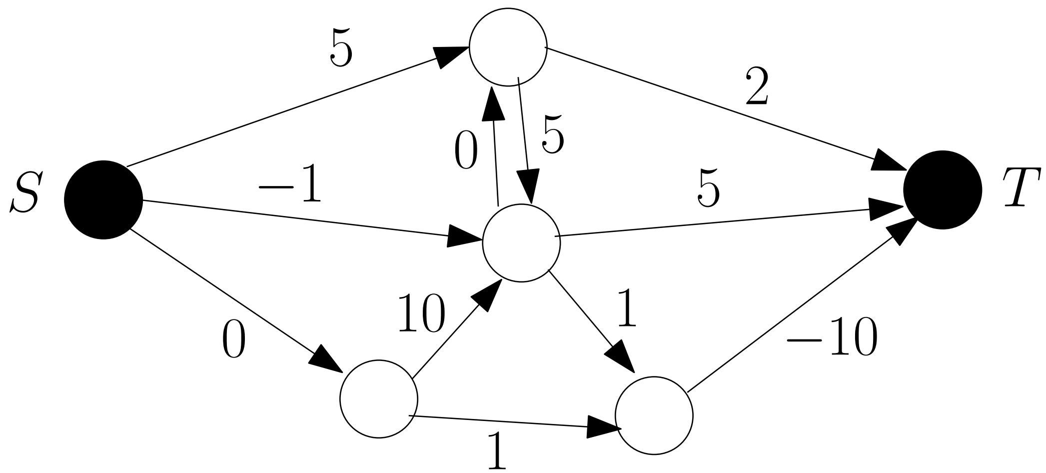


Application: Find important clusters in a database.

Problem Versions

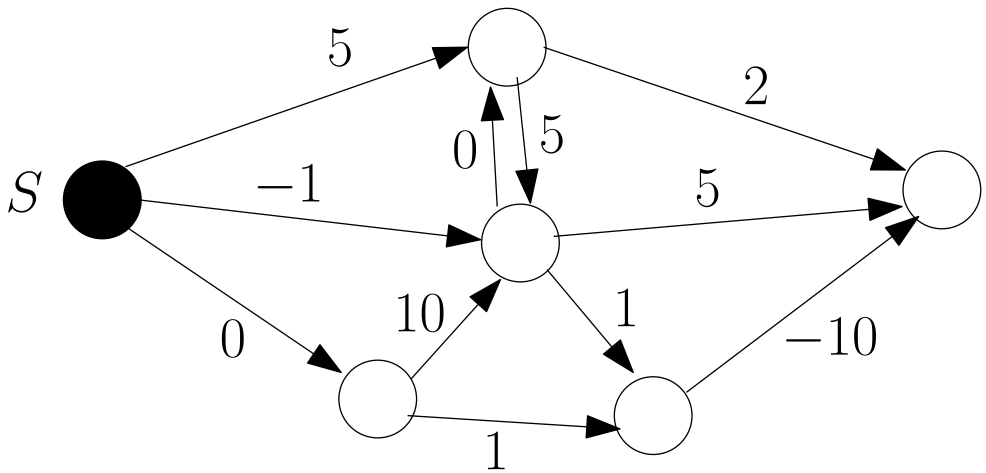
Problem Versions

Single Source, Single Destination (S-T Best Path)



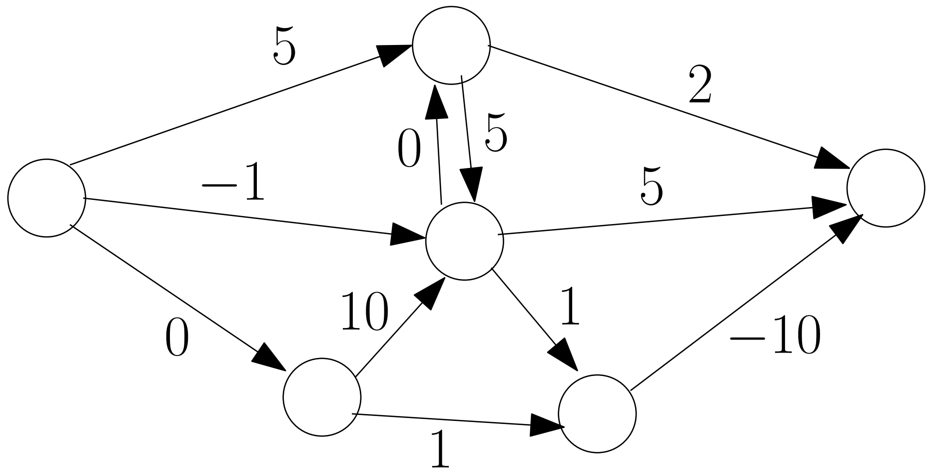
Problem Versions

Single Source (every destination) Best Path



Problem Versions

All Pairs Best Path



Talk Outline

1. Algorithms for All Pairs Best Paths

2. Algorithms for Single Source Best Paths

3. Directions for Further Research

All Pairs Path Problems – Results

n-number of vertices, m-number of edges

Problem	Previous Best	Our Results
AP Max Triangle	n^3	$n^{2.58}$ (VWY 2006)
AP Max Bottleneck Paths	n^3	$n^{2.79}$ (VWY 2007)
AP Min Nondecreasing Paths	n^3	$n^{2.9}$ (V 2008)
k Bits of Distance Product	$n^3/\log n$ (Chan 2005)	$2^k n^{2.69}$ (VW 2006).

All Pairs Path Problems – Outline

1. Path Problems and Matrix Products
2. Properties and Algorithms
3. Techniques
4. Example
5. Summary of results

Path Problems and $\otimes$ Products

For all these problems, the length of $v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k$ is

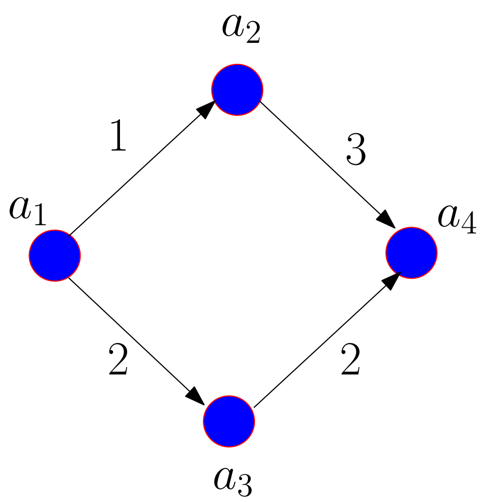
$$\ell(v_1, \ldots, v_k) = \ell(v_1, \ldots, v_{k-1}) \otimes w(v_{k-1}, v_k), \;\; \ell(v_1, v_2) = w(v_1, v_2),$$

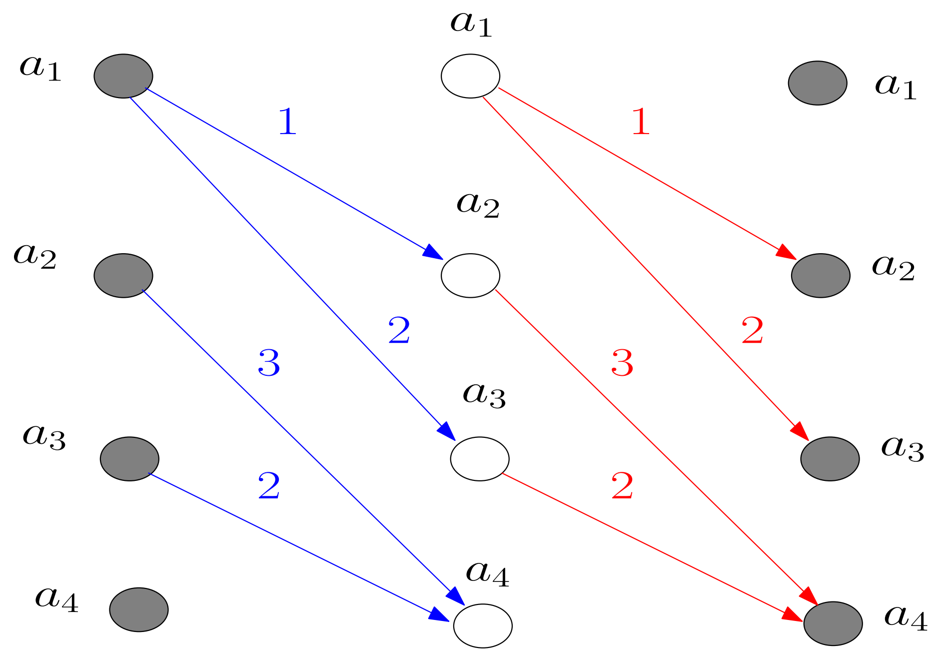
where $\otimes$ is different for each problem.

Problem	$\otimes$
Shortest Paths	$+$
Bottleneck Paths	$\min$
Nondecreasing Paths	$\leq'$

$a \leq' b$ returns b if $a \leq b$ and ∞ otherwise.

Path Problems and Matrices





$$\begin{pmatrix} \infty & 1 & 2 & \infty \\ \infty & \infty & \infty & 3 \\ \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty \end{pmatrix}$$

shortest paths

$$\begin{pmatrix} -\infty & 1 & 2 & -\infty \\ -\infty & -\infty & -\infty & 3 \\ -\infty & -\infty & -\infty & 2 \\ -\infty & -\infty & -\infty & -\infty \end{pmatrix}$$

bottleneck paths

$$\begin{pmatrix} \varepsilon_0 & 1 & 2 & \varepsilon_0 \\ \varepsilon_0 & \varepsilon_0 & \varepsilon_0 & 3 \\ \varepsilon_0 & \varepsilon_0 & \varepsilon_0 & 2 \\ \varepsilon_0 & \varepsilon_0 & \varepsilon_0 & \varepsilon_0 \end{pmatrix}$$

in general

Path Problems and Matrix Products

Definition [$\{\oplus, \otimes\}$-Product]:

Given two $n \times n$ real matrices A and B, and two operations $\otimes$ and $\oplus$ on $\mathbb{R}$, such that $\oplus$ is commutative and associative, the $\{\oplus, \otimes\}$-product of A and B is the $n \times n$ matrix C given by

$$C[i,j] := \bigoplus_{k=1}^{n} (A[i,k] \otimes B[k,j]), \forall i,j = 1, \ldots, n.$$

For our path problems, $\oplus$ is always $\max$ or $\min$.

Various Matrix Products: definitions

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Dominance Product:

$$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$$

Various Matrix Products: definitions

Algebraic Product:

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Dominance Product:

$$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$$

$(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product: (CoppersmithWinograd90)

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$ $n^\omega = n^{2.376}$

Distance Product:

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$

MaxMin Product:

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$

Dominance Product:

$C[i,j] = (A \odot B)[i,j] = |\{k \ : \ A[i,k] \leq B[k,j]\}|.$

$(\min, \leq)$-Product:

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \ A[i,k] \leq B[k,j]\}.$

Various Matrix Products: definitions

Algebraic Product: (CoppersmithWinograd90)

$$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}. \quad n^{\omega} = n^{2.376}$$

Distance Product:

$$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$$

MaxMin Product:

$$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$$

Dominance Product: (Matousek91)

$$C[i,j] = (A \odot B)[i,j] = |\{k \; : \; A[i,k] \leq B[k,j]\}|. \quad n^{\frac{3+\omega}{2}}$$

$(\min, \leq)$-Product:

$$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \; A[i,k] \leq B[k,j]\}.$$

Various Matrix Products: definitions

Algebraic Product: (CoppersmithWinograd90)

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}$. $n^\omega = n^{2.376}$

Distance Product:

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}$.

MaxMin Product: (VWY07)

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}$. $n^{2+\frac{\omega}{3}}$

Dominance Product: (Matousek91)

$C[i,j] = (A \odot B)[i,j] = |\{k \, : \, A[i,k] \leq B[k,j]\}|$. $n^{\frac{3+\omega}{2}}$

$(\min, \leq)$-Product: (VWY07, V08)

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \, A[i,k] \leq B[k,j]\}$. $n^{2+\frac{\omega}{3}}$

Various Matrix Products: definitions

Algebraic Product: (CoppersmithWinograd90)

$C[i,j] = (A \cdot B)[i,j] = \sum_k \{A[i,k] \cdot B[k,j]\}.$ $n^\omega = n^{2.376}$

Distance Product: (Chan07)

$C[i,j] = (A \star B)[i,j] = \min_k \{A[i,k] + B[k,j]\}.$ $n^3 / \log^2 n$

MaxMin Product: (VWY07)

$C[i,j] = (A \bullet B)[i,j] = \max_k \min\{A[i,k], B[k,j]\}.$ $n^{2+\frac{\omega}{3}}$

Dominance Product: (Matousek91)

$C[i,j] = (A \odot B)[i,j] = |\{k \, : \, A[i,k] \leq B[k,j]\}|.$ $n^{\frac{3+\omega}{2}}$

$(\min, \leq)$-Product: (VWY07, V08)

$C[i,j] = (A \otimes B)[i,j] = \min_k \{B[k,j] : \, A[i,k] \leq B[k,j]\}.$ $n^{2+\frac{\omega}{3}}$

Properties and Path Algorithms

Let $OPT[x, y]$ be the best $x \longrightarrow y$ path length. $OPT[x, x] = \varepsilon_1$.

Edge-Padding Property:

There are operations $\oplus$ and $\otimes$, such that $\oplus$ is commutative and associative, and for all pairs of vertices x, y in the graph,

$$OPT[x, y] = \bigoplus_{z \in V} (OPT[x, z] \otimes w(z, y)).$$

Properties and Path Algorithms

Let $OPT[x, y]$ be the best $x \longrightarrow y$ path length. $OPT[x, x] = \varepsilon_1$.

Edge-Padding Property:

There are operations $\oplus$ and $\otimes$, such that $\oplus$ is commutative and associative, and for all pairs of vertices x, y in the graph,

$$OPT[x, y] = \bigoplus_{z \in V} (OPT[x, z] \otimes w(z, y)).$$

This holds when $\otimes$ is right-distributive over $\oplus$.

Properties and Path Algorithms Cont.

Midpoint Property:

If $x \longrightarrow v_1 \longrightarrow \ldots \longrightarrow v_k \longrightarrow y$ is an optimal path, then for all i, $x \longrightarrow \ldots \longrightarrow v_i$ and $v_i \longrightarrow \ldots \longrightarrow y$ are also optimal.

Properties and Path Algorithms Cont.

Midpoint Property:

If $x \longrightarrow v_1 \longrightarrow \ldots \longrightarrow v_k \longrightarrow y$ is an optimal path, then for all i,

$x \longrightarrow \ldots \longrightarrow v_i$ and $v_i \longrightarrow \ldots \longrightarrow y$ are also optimal.

There are operations $\oplus$ and $\otimes$, where $\oplus$ is associative and commutative, so that for all pairs of vertices x, y in the graph,

$$OPT[x,y] = \bigoplus_{z \in V} (OPT[x,z] \otimes OPT[z,y]).$$

Properties and Path Algorithms Cont.

Midpoint Property:

If $x \longrightarrow v_1 \longrightarrow \ldots \longrightarrow v_k \longrightarrow y$ is an optimal path, then for all i,

$x \longrightarrow \ldots \longrightarrow v_i$ and $v_i \longrightarrow \ldots \longrightarrow y$ are also optimal.

There are operations $\oplus$ and $\otimes$, where $\oplus$ is associative and commutative,

so that for all pairs of vertices x, y in the graph,

$$OPT[x,y] = \bigoplus_{z \in V} (OPT[x,z] \otimes OPT[z,y]).$$

This holds when $\otimes$ is associative and distributes over $\oplus$.

Properties and Path Algorithms Cont.

When optimal paths have length at most N:

Properties and Path Algorithms Cont.

When optimal paths have length at most N:

If the edge-padding property holds, all pairs best paths can be done in $O(N \times T[(\oplus, \otimes)\text{-product}])$.

Properties and Path Algorithms Cont.

When optimal paths have length at most N:

If the edge-padding property holds, all pairs best paths can be done in $O(N \times T[(\oplus, \otimes)\text{-product}])$.

If the midpoint property holds, all pairs best paths can be done in $O(\log N \times T[(\oplus, \otimes)\text{-product}])$.

Properties and Path Algorithms Cont.

When optimal paths have length at most N:

If the edge-padding property holds, all pairs best paths can be done in $O(N \times T[(\oplus, \otimes)\text{-product}])$.

If the midpoint property holds, all pairs best paths can be done in $O(\log N \times T[(\oplus, \otimes)\text{-product}])$.

When $(\mathbb{R}, \oplus, \otimes, \varepsilon_0, \varepsilon_1)$ form a semiring, then all pairs best paths can be done in $O(T[(\oplus, \otimes)\text{-product}])$ (AHU74).

Matrix Products and Path Algorithms

Matrix Products and Path Algorithms

AP Shortest Paths and AP Bottleneck Paths are problems over semirings.

Matrix Products and Path Algorithms

AP Shortest Paths and AP Bottleneck Paths are problems over semirings.

AP Min Nondecreasing Paths does not even have the midpoint property.

$\leq$ is not associative: $[(2 \leq 1) \leq 3] = \infty$ and $[2 \leq (1 \leq 3)] = 3$.

Matrix Products and Path Algorithms

AP Shortest Paths and AP Bottleneck Paths are problems over semirings.

AP Min Nondecreasing Paths does not even have the midpoint property.

$\leq$ is not associative: $[(2 \leq 1) \leq 3] = \infty$ and $[2 \leq (1 \leq 3)] = 3$.

Still, we can show that if $(\min, \leq)$ product is in subcubic time, so is AP Minimum Nondecreasing Paths. *(short paths - long paths method)*

Matrix Products and Path Algorithms

AP Shortest Paths and AP Bottleneck Paths are problems over semirings.

AP Min Nondecreasing Paths does not even have the midpoint property.

$\leq$ is not associative: $[(2 \leq 1) \leq 3] = \infty$ and $[2 \leq (1 \leq 3)] = 3$.

Still, we can show that if $(\min, \leq)$ product is in subcubic time, so is AP Minimum Nondecreasing Paths. *(short paths - long paths method)*

Hence, we can concentrate on matrix products.

Techniques for Computing Matrix Products

Techniques for Computing Matrix Products

1. **Bucketting:** Preprocess each input matrix and assign its entries in a 1-to-1 fashion to some number of buckets, each getting a small number of entries. For each input matrix X and each bucket b, create a new matrix X_b.

Techniques for Computing Matrix Products

1. **Bucketting:** Preprocess each input matrix and assign its entries in a 1-to-1 fashion to some number of buckets, each getting a small number of entries. For each input matrix X and each bucket b, create a new matrix X_b.

2. **Bucket Processing:** For each bucket b, multiply A_b and B_b using (perhaps recursively) a different matrix product $(\oplus', \otimes')$.

Techniques for Computing Matrix Products

1. **Bucketting:** Preprocess each input matrix and assign its entries in a 1-to-1 fashion to some number of buckets, each getting a small number of entries. For each input matrix X and each bucket b, create a new matrix X_b.

2. **Bucket Processing:** For each bucket b, multiply A_b and B_b using (perhaps recursively) a different matrix product $(\oplus', \otimes')$.

3. **Exhaustive Search:** The bucket processing step provides information which allows us to choose a small number of buckets on which the problem is solved by exhaustive search.

Example: $(\min, \leq)$-Product

We want $a_{ij} = \min_k\{B[k,j] \mid A[i,k] \leq B[k,j]\}$.

We will use the dominance product: $(A \odot B)[i,j] = |\{k \; : \; A[i,k] \leq B[k,j]\}|$.

Example: $(\min, \leq)$-Product

We want $a_{ij} = \min_k\{B[k,j] \mid A[i,k] \leq B[k,j]\}$.

We will use the dominance product: $(A \odot B)[i,j] = |\{k \,:\, A[i,k] \leq B[k,j]\}|$.

1. Take the columns of B and sort the entries of each column.

$$B = \begin{pmatrix} 10 & 2 & 0 & 7 \\ -1.1 & 3 & -1 & 2.1 \\ 5.1 & 7 & -2 & 4 \\ 3.2 & 1 & -3 & 2.1 \end{pmatrix}$$

column 1 : $A[2,1], \quad A[4,1], \quad A[3,1], \quad A[1,1]$

column 2 : $A[4,2], \quad A[1,2], \quad A[2,2], \quad A[3,2]$

column 3 : $A[5,3], \quad A[3,3], \quad A[2,3], \quad A[1,3]$

column 4 : $A[4,4], \quad A[2,4], \quad A[3,4], \quad A[1,4]$

Example: $(\min, \leq)$-Product

We want $a_{ij} = \min_k\{B[k,j] \mid A[i,k] \leq B[k,j]\}$.

We will use the dominance product: $(A \odot B)[i,j] = |\{k : A[i,k] \leq B[k,j]\}|$.

1. Take the columns of B and sort the entries of each column.
2. (Technique 1) Bucket the entries of each column of B, in their sorted order into s roughly equal buckets.

$$B = \begin{pmatrix} 10 & 2 & 0 & 7 \\ -1.1 & 3 & -1 & 2.1 \\ 5.1 & 7 & -2 & 4 \\ 3.2 & 1 & -3 & 2.1 \end{pmatrix}$$

column 1 : $A[2,1]$, $A[4,1]$, $A[3,1]$, $A[1,1]$

column 2 : $A[4,2]$, $A[1,2]$, $A[2,2]$, $A[3,2]$

column 3 : $A[5,3]$, $A[3,3]$, $A[2,3]$, $A[1,3]$

column 4 : $A[4,4]$, $A[2,4]$, $A[3,4]$, $A[1,4]$

Example: $(\min, \leq)$-Product cont.

3. (Technique 1 - **Bucketting**) For each bucket b create a matrix $B(b)$ containing only the elements in bucket b and $-\infty$ in all other entries.

$$B(1) = \begin{pmatrix} -\infty & 2 & -\infty & -\infty \\ -1.1 & -\infty & -\infty & 2.1 \\ -\infty & -\infty & -2 & -\infty \\ 3.2 & 1 & -3 & 2.1 \end{pmatrix} \quad B(2) = \begin{pmatrix} 10 & -\infty & 0 & 7 \\ -\infty & 3 & -1 & -\infty \\ 5.1 & 7 & -\infty & 4 \\ -\infty & -\infty & -\infty & -\infty \end{pmatrix}$$

Example: $(\min, \leq)$-Product cont.

We want $a_{ij} = \min_k \{B[k,j] \mid A[i,k] \leq B[k,j]\}$.

We will use the dominance product: $(A \odot B)[i,j] = |\{k \,:\, A[i,k] \leq B[k,j]\}|$.

4. (Technique 2 - **Bucket Processing**) Compute $A \odot B(b)$ for each b.

$B \odot B(2) =$

$$\begin{pmatrix} 10 & 2 & 0 & 7 \\ -1.1 & 3 & -1 & 2.1 \\ 5.1 & 7 & -2 & 4 \\ 3.2 & 1 & -3 & 2.1 \end{pmatrix} \odot \begin{pmatrix} 10 & -\infty & 0 & 7 \\ -\infty & 3 & -1 & -\infty \\ 5.1 & 7 & -\infty & 4 \\ -\infty & -\infty & -\infty & -\infty \end{pmatrix} = \begin{pmatrix} 2 & 2 & 0 & 1 \\ 2 & 2 & 1 & 2 \\ 2 & 1 & 0 & 2 \\ 2 & 2 & 0 & 2 \end{pmatrix}$$

This tells us for every bucket b and each i, j, the number of coords k such that $B[k,j]$ is in bucket b and $A[i,k] \leq B[k,j]$.

This step takes $O(sn^{\frac{3+\omega}{2}})$ since dominance product takes $O(n^{\frac{3+\omega}{2}})$.

Example: $(\min, \leq)$-Product cont.

Example: $(\min, \leq)$-Product cont.

5. For each i, j we know the smallest bucket b in which there is an entry $B[k, j]$ such that $A[i, k] \leq B[k, j]$.

Example: $(\min, \leq)$-Product cont.

5. For each i, j we know the smallest bucket b in which there is an entry $B[k, j]$ such that $A[i, k] \leq B[k, j]$.

6. (Tecnique 3 - **Exhaustive Search**) For each i, j, search that bucket for smallest $B[k, j]$ - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

Example: $(\min, \leq)$-Product cont.

5. For each i, j we know the smallest bucket b in which there is an entry $B[k, j]$ such that $A[i, k] \leq B[k, j]$.

6. (Tecnique 3 - **Exhaustive Search**) For each i, j, search that bucket for smallest $B[k, j]$ - there are at most $O(n/s)$ entries we have to go through for each pair i, j.

   This step takes $O(n^3/s)$ and explicitly finds witnesses.

7. The overall runtime is minimized for $s = n^{\frac{3-\omega}{4}}$ and the runtime is then $O(n^{\frac{9+\omega}{4}}) = O(n^{2.85})$.

Summary of Results

Summary of Results

All Pairs Maximum Node Weighted Triangles - either by using dominance product, or by using rectangular (algebraic) matrix multiplication. Best Running Time: $O(n^{2.575})$. (VW STOC06, VWY ICALP06)

Summary of Results

All Pairs Maximum Node Weighted Triangles - either by using dominance product, or by using rectangular (algebraic) matrix multiplication. Best Running Time: $O(n^{2.575})$. (VW STOC06, VWY ICALP06)

All Pairs Shortest Paths - compute k most significant bits of the distance product in $O(2^k n^{2.688})$ time using dominance product. (VW STOC06)

Summary of Results

All Pairs Maximum Node Weighted Triangles - either by using dominance product, or by using rectangular (algebraic) matrix multiplication. Best Running Time: $O(n^{2.575})$. (VW STOC06, VWY ICALP06)

All Pairs Shortest Paths - compute k most significant bits of the distance product in $O(2^k n^{2.688})$ time using dominance product. (VW STOC06)

All Pairs Minimum Nondecreasing Paths - compute $(\min, \leq)$-Product using dominance product, and then apply short path - long path technique of Zwick/Chan. Best Running Time: $O(n^{2.896})$. (V SODA08)

Summary of Results

All Pairs Maximum Node Weighted Triangles - either by using dominance product, or by using rectangular (algebraic) matrix multiplication. Best Running Time: $O(n^{2.575})$. (VW STOC06, VWY ICALP06)

All Pairs Shortest Paths - compute k most significant bits of the distance product in $O(2^k n^{2.688})$ time using dominance product. (VW STOC06)

All Pairs Minimum Nondecreasing Paths - compute $(\min, \leq)$-Product using dominance product, and then apply short path - long path technique of Zwick/Chan. Best Running Time: $O(n^{2.896})$. (V SODA08)

All Pairs Bottleneck Paths - compute $(\max, \min)$-Product using $(\min, \leq)$-Product. Best Running Time: $O(n^{2.792})$. (VWY STOC07)

Single Source Path Problems

Single Source Path Problems

Single Source Shortest Paths - $O(m+n)$ time algorithm (Thorup) known for undirected graphs in the RAM machine model. No linear time algorithm known for directed graphs.

Single Source Path Problems

Single Source Shortest Paths - $O(m+n)$ time algorithm (Thorup) known for undirected graphs in the RAM machine model. No linear time algorithm known for directed graphs.

Single Source Nondecreasing Paths - Previously, the best algorithm was Dijkstra's algorithm, with Fibonacci Heaps, $O(m + n \log n)$. We give the first $O(m+n)$ algorithm in the RAM model. In the pointer machine model we give a $O(m \log \log n)$ algorithm. (V08)

Single Source Path Problems

Single Source Shortest Paths - $O(m+n)$ time algorithm (Thorup) known for undirected graphs in the RAM machine model. No linear time algorithm known for directed graphs.

Single Source Nondecreasing Paths - Previously, the best algorithm was Dijkstra's algorithm, with Fibonacci Heaps, $O(m+n\log n)$. We give the first $O(m+n)$ algorithm in the RAM model. In the pointer machine model we give a $O(m\log\log n)$ algorithm. (V08)

Single Source Bottleneck Paths - A $O(m+n)$ algorithm is known for the undirected single source, single target version (Punnen91). In the general case: Dijkstra's with Fibonacci Heaps $O(m+n\log n)$.

Directions for Further Research

1. More single source algorithms - better single source algorithm for bottleneck paths?
2. Parallel Algorithms
3. Combinatorial Algorithms

Parallel Algorithms

Our all pairs algorithms use fast matrix multiplication and techniques involving sorting, bucketting and exhaustive search.

Parallel Algorithms

Our all pairs algorithms use fast matrix multiplication and techniques involving sorting, bucketting and exhaustive search.

Fast Matrix Multiplication can be done in parallel on $O(n^\omega)$ processors and $O(\mathrm{poly}\log \mathrm{n})$ time.

Parallel Algorithms

Our all pairs algorithms use fast matrix multiplication and techniques involving sorting, bucketting and exhaustive search.

Fast Matrix Multiplication can be done in parallel on $O(n^\omega)$ processors and $O(\mathrm{poly}\log\mathrm{n})$ time.

Hence our all pairs algorithms running sequentially in $O(n^c)$ time can be done in parallel on $O(n^c)$ processors and $O(\mathrm{poly}\log\mathrm{n})$ time.

Parallel Algorithms Cont.

On graphs with small separators of size $s(n)$:

Parallel Algorithms Cont.

On graphs with small separators of size $s(n)$:

Pan and Reif 1989: all pairs path problems on semirings in $O(\log^2 n)$ parallel time and using $O(mn/\log n + ns^2(n))$ processors.

Parallel Algorithms Cont.

On graphs with small separators of size $s(n)$:

Pan and Reif 1989: all pairs path problems on semirings in $O(\log^2 n)$ parallel time and using $O(mn/\log n + ns^2(n))$ processors.

Cohen 1993: K-source shortest paths in $O(\log^2 n)$ parallel time using $O(K(n + s^2(n)))$ work, after $O(\log^2 n)$–time, $O((n + s(n)^3) \log n)$–work preprocessing.

Parallel Algorithms Cont.

On graphs with small separators of size $s(n)$:

Pan and Reif 1989: all pairs path problems on semirings in $O(\log^2 n)$ parallel time and using $O(mn/\log n + ns^2(n))$ processors.

Cohen 1993: K-source shortest paths in $O(\log^2 n)$ parallel time using $O(K(n + s^2(n)))$ work, after $O(\log^2 n)$–time, $O((n + s(n)^3)\log n)$–work preprocessing.

Conjecture: for K-source bottleneck / nondecreasing paths reduce preprocessing work to $O((n + s(n)^\alpha)\log n)$–work where:

$\alpha = 2.792$ for bottleneck and $\alpha = 2.896$ for nondecreasing paths.

Purely Combinatorial Algorithms

Purely combinatorial – nonalgebraic, nonsubtractive.

Purely Combinatorial Algorithms

Purely combinatorial – nonalgebraic, nonsubtractive.

Examples: Four Russians Algorithm for Matrix Multiplication in $O(n^3/\log^2 n)$, Chan algorithm for all pairs shortest paths in $O(n^3 \log\log^3 n/\log^2 n)$.

Purely Combinatorial Algorithms

Purely combinatorial – nonalgebraic, nonsubtractive.

Examples: Four Russians Algorithm for Matrix Multiplication in $O(n^3/\log^2 n)$, Chan algorithm for all pairs shortest paths in $O(n^3 \log\log^3 n/\log^2 n)$.

We want similar runtimes for AP bottleneck paths, and AP nondecreasing paths.

Purely Combinatorial Algorithms Cont.

Purely Combinatorial Algorithms Cont.

The somewhat sparse case – $O(mn \log(n^2/m)/\log^2 n)$ algorithm for matrix multiplication, transitive closure and max weight triangle (BVW 08).

Previous best was $O(mn/\log n)$ (Chan06). Ours is better when $m = n^{2-o(1)}$.

Purely Combinatorial Algorithms Cont.

The somewhat sparse case – $O(mn \log(n^2/m)/\log^2 n)$ algorithm for matrix multiplication, transitive closure and max weight triangle (BVW 08).

Previous best was $O(mn/\log n)$ (Chan06). Ours is better when $m = n^{2-o(1)}$.

Similar algorithms for AP shortest, AP nondecreasing, or AP bottleneck paths?

Approximate Timeline

Approximate Timeline

- November 19th - STOC deadline - combinatorial results?

Approximate Timeline

- November 19th - STOC deadline - combinatorial results?
- After Thanksgiving - start writing thesis

Approximate Timeline

- November 19th - STOC deadline - combinatorial results?
- After Thanksgiving - start writing thesis
- Beginning of March - have all results until now merged

Approximate Timeline

- November 19th - STOC deadline - combinatorial results?
- After Thanksgiving - start writing thesis
- Beginning of March - have all results until now merged
- April - thesis defense?

Thank you!