Approximating All-Pair Bounded-Leg Shortest Path and APSP-AF in Truly-Subcubic Time

Ran Duan, Hanlin Ren

Institute for Interdisciplinary Information Sciences
Tsinghua University

July 12, 2018

Outline

1. Introduction
   - Motivation
   - Our results
2. The algorithm
   - 1. Exact product for small distances
   - 2. Approximate product for arbitrary distances
   - 3. Main procedure
3. (Sketch of) a faster algorithm
4. Conclusions and open problems

Bounded-leg shortest path

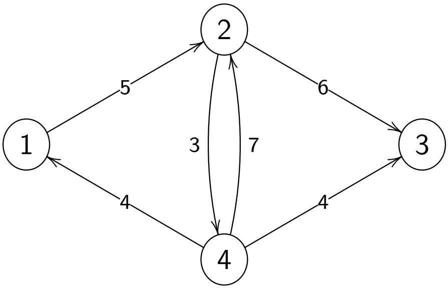
- Motivation 1: the bounded-leg shortest path problem
- Given a weighted directed graph, answer the following queries:

Bounded-leg shortest path

- Motivation 1: the bounded-leg shortest path problem
- Given a weighted directed graph, answer the following queries:
  - what's the shortest path from s to t, if only edges of length $\leq L$ are considered?

Bounded-leg shortest path

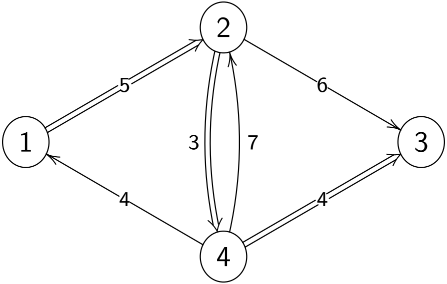
- Motivation 1: the bounded-leg shortest path problem
- Given a weighted directed graph, answer the following queries:
  - what's the shortest path from s to t, if only edges of length $\leq L$ are considered?



A graph G.

Bounded-leg shortest path

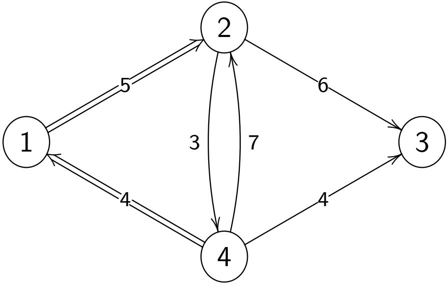
- Motivation 1: the bounded-leg shortest path problem
- Given a weighted directed graph, answer the following queries:
    - what's the shortest path from s to t, if only edges of length $\leq L$ are considered?



$Query(s = 1, t = 3, L = 5)$

Bounded-leg shortest path

- Motivation 1: the bounded-leg shortest path problem
- Given a weighted directed graph, answer the following queries:
  - what's the shortest path from s to t, if only edges of length $\leq L$ are considered?



$Query(s = 4, t = 2, L = 6)$

APSP-AF

- Motivation 2: the APSP-AF problem.
  - **A**ll-**P**air-**S**hortest-**P**ath for **A**ll **F**lows.

APSP-AF

- Motivation 2: the APSP-AF problem.
  - **A**ll-**P**air-**S**hortest-**P**ath for **A**ll **F**lows.
- Given a graph, in which each edge has a *length* l and a *capacity* c, answer the following queries:
  - what's the shortest path from s to t, if only edges of capacity $\geq f$ are considered?

APSP-AF

- Motivation 2: the APSP-AF problem.
  - **A**ll-**P**air-**S**hortest-**P**ath for **A**ll **F**lows.
- Given a graph, in which each edge has a *length* l and a *capacity* c, answer the following queries:
  - what's the shortest path from s to t, if only edges of capacity $\geq f$ are considered?
- A generalization of bounded-leg shortest path.

Our results

- A simple algorithm for $(1+\epsilon)$-approximating APSP-AF in $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W)$ preprocessing time[1] and $O(\log\frac{\log(nW)}{\epsilon})$ query time
  - where W is the maximum edge length,
  - and $\omega < 2.373$ is the matrix-multiplication exponent.

[1] $\tilde{O}$ hides $\mathrm{polylog}(n)$ factors
[2] $O(n^{3-\delta}\mathrm{polylog}(\epsilon, W))$ for some $\delta > 0$

Our results

- A simple algorithm for $(1+\epsilon)$-approximating APSP-AF in $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W)$ preprocessing time[1] and $O(\log\frac{\log(nW)}{\epsilon})$ query time
  - where W is the maximum edge length,
  - and $\omega < 2.373$ is the matrix-multiplication exponent.
- An algorithm in $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$ preprocessing time

[1] $\tilde{O}$ hides $\mathrm{polylog}(n)$ factors

[2] $O(n^{3-\delta}\mathrm{polylog}(\epsilon, W))$ for some $\delta > 0$

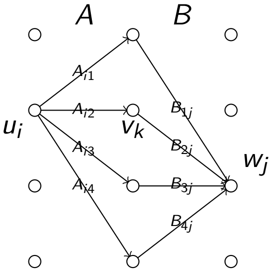
Our results

- A simple algorithm for $(1+\epsilon)$-approximating APSP-AF in $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W)$ preprocessing time[1] and $O(\log\frac{\log(nW)}{\epsilon})$ query time
  - where W is the maximum edge length,
  - and $\omega < 2.373$ is the matrix-multiplication exponent.
- An algorithm in $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$ preprocessing time
- This is the first truly-subcubic[2] time algorithm for such problems.

[1] $\tilde{O}$ hides $\mathrm{polylog}(n)$ factors
[2] $O(n^{3-\delta}\mathrm{polylog}(\epsilon, W))$ for some $\delta > 0$

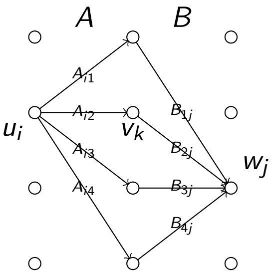
Various Matrix Products

- Distance product: $(A \star B)_{ij} = \min_k\{A_{ik} + B_{kj}\}$.



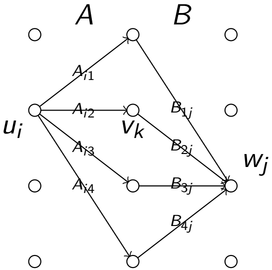
Various Matrix Products

- Distance product: $(A \star B)_{ij} = \min_k\{A_{ik} + B_{kj}\}$.
  - The shortest path from u_i to w_j.



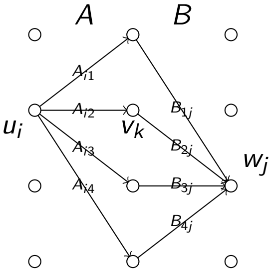
Various Matrix Products

- Distance product: $(A \star B)_{ij} = \min_k\{A_{ik} + B_{kj}\}$.
  - The shortest path from u_i to w_j.
- Max-min product:
  $(A \bigtriangledown B)_{ij} = \max_k \min\{A_{ik}, B_{kj}\}$.



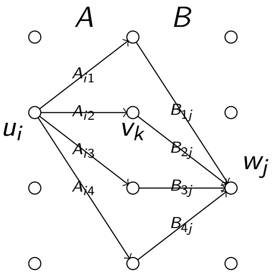
Various Matrix Products

- Distance product: $(A \star B)_{ij} = \min_k\{A_{ik} + B_{kj}\}$.
  - The shortest path from u_i to w_j.
- Max-min product:
  $(A \circledvee B)_{ij} = \max_k \min\{A_{ik}, B_{kj}\}$.
  - The maximum flow that can be pushed from u_i to w_j.



Various Matrix Products

- Distance product: $(A \star B)_{ij} = \min_k\{A_{ik} + B_{kj}\}$.
  - The shortest path from u_i to w_j.
- Max-min product:
  $(A \circledvee B)_{ij} = \max_k \min\{A_{ik}, B_{kj}\}$.
  - The maximum flow that can be pushed from u_i to w_j.
  - $A \circledvee B$ can be computed in $O(n^{\frac{3+\omega}{2}})$ time [Duan and Pettie, 2009].



Query Time

- We compute a matrix whose entries A_{ij} are sets of (d, f) pairs.
  - $A_{ij} = \{(d_k, f_k)\}$
  - Intuitively, an entry $(d, f) \in A_{ij}$ indicates a path from i to j, with minimum capacity f and distance $\approx d$.

Query Time

- We compute a matrix whose entries A_{ij} are sets of (d, f) pairs.
  - $A_{ij} = \{(d_k, f_k)\}$
  - Intuitively, an entry $(d, f) \in A_{ij}$ indicates a path from i to j, with minimum capacity f and distance $\approx d$.
  - We call such a matrix "*df*-matrix".

Query Time

- We compute a matrix whose entries A_{ij} are sets of (d, f) pairs.
  - $A_{ij} = \{(d_k, f_k)\}$
  - Intuitively, an entry $(d, f) \in A_{ij}$ indicates a path from i to j, with minimum capacity f and distance $\approx d$.
  - We call such a matrix "df-matrix".
- To answer a query (s, t, f), simply find

$$\min\{d : (d, f') \in A_{st}, f' \geq f\}.$$

Query Time

- We compute a matrix whose entries A_{ij} are sets of (d, f) pairs.
  - $A_{ij} = \{(d_k, f_k)\}$
  - Intuitively, an entry $(d, f) \in A_{ij}$ indicates a path from i to j, with minimum capacity f and distance $\approx d$.
  - We call such a matrix "df-matrix".
- To answer a query (s, t, f), simply find

$$\min\{d : (d, f') \in A_{st}, f' \geq f\}.$$

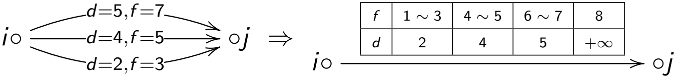
- Query time $O(\log |A_{st}|)$, and $|A_{st}| = O(\frac{\log(nW)}{\epsilon})$.

$A(f)$

- For a df-matrix A and a flow f, define $A(f)$ be the matrix satisfying $A(f)_{ij} = \min\{d : \exists(d, f') \in A_{ij}, f' \geq f\}$.
  - Given flow constraint f, what's the best path in A_{ij}?

$A(f)$

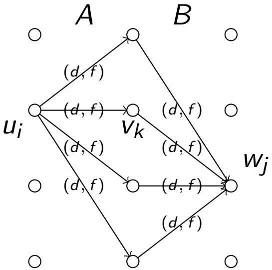
- For a df-matrix A and a flow f, define $A(f)$ be the matrix satisfying $A(f)_{ij} = \min\{d : \exists(d, f') \in A_{ij}, f' \geq f\}$.
  - Given flow constraint f, what's the best path in A_{ij}?



f	$1 \sim 3$	$4 \sim 5$	$6 \sim 7$	8
d	2	4	5	$+\infty$

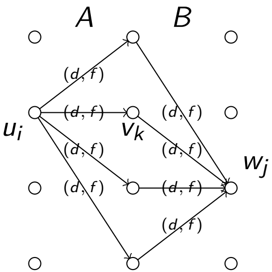
1. Exact product for small distances

- Given two df-matrices A, B, where **all distances** $d \leq R$ and R is a small number.



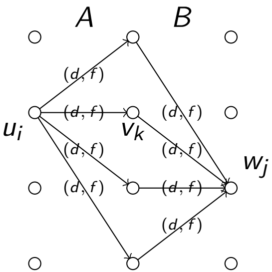
1. Exact product for small distances

- Given two df-matrices A, B, where **all distances** $d \le R$ and R is a small number.
- We'd like to compute some df-matrix C, such that $\forall f, C(f) = A(f) \star B(f)$.
    - Recall $\star$ is the distance product.



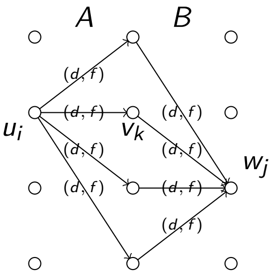
1. Exact product for small distances

- Given two df-matrices A, B, where **all distances** $d \le R$ and R is a small number.
- We'd like to compute some df-matrix C, such that $\forall f, C(f) = A(f) \star B(f)$.
  - Recall $\star$ is the distance product.
  - Also let's say $C = A \star B$ for brevity.



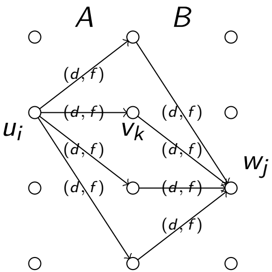
1. Exact product for small distances

- Given two df-matrices A, B, where **all distances** $d \leq R$ and R is a small number.
- We'd like to compute some df-matrix C, such that $\forall f, C(f) = A(f) \star B(f)$.
  - Recall $\star$ is the distance product.
  - Also let's say $C = A \star B$ for brevity.
- This takes $O(n^{\frac{3+\omega}{2}} R^2)$ time.



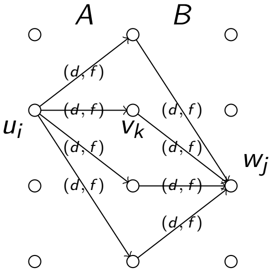
1. Exact product for small distances

- Given d, what's the largest f s.t. $C_{ij}(f) \leq d$?



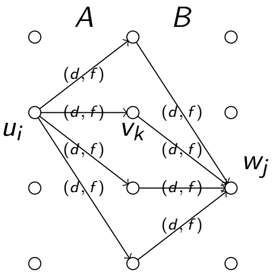
1. Exact product for small distances

- Given $d_1 + d_2 = d$, what's the largest f s.t. $A_{ij}(f) \le d_1$ and $B_{ij}(f) \le d_2$?
  -



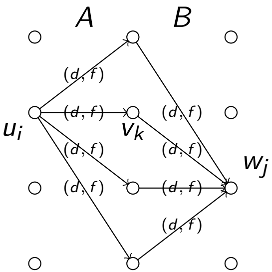
1. Exact product for small distances

- Given $d_1 + d_2 = d$, what's the largest f s.t. $A_{ij}(f) \leq d_1$ and $B_{ij}(f) \leq d_2$?
    -
- Let's define $A_{ij}^{(d)} = \max\{f : (d', f) \in A_{ij}, d' \leq d\}$.



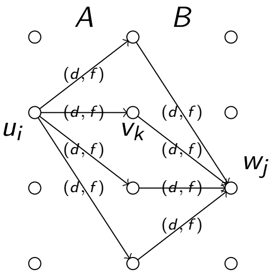
1. Exact product for small distances

- Given $d_1 + d_2 = d$, what's the largest f s.t. $A_{ij}(f) \le d_1$ and $B_{ij}(f) \le d_2$?
  - $\max_k \min\{A_{ik}^{(d_1)}, B_{kj}^{(d_2)}\}$
- Let's define $A_{ij}^{(d)} = \max\{f : (d', f) \in A_{ij}, d' \le d\}$.



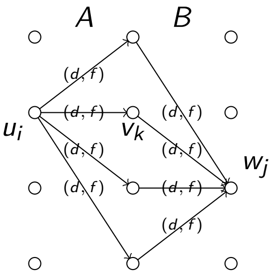
1. Exact product for small distances

- Given d, what's the largest f s.t. $C_{ij}(f) \le d$?
  - $\max_{d_1+d_2=d} \max_k \min\{A_{ik}^{(d_1)}, B_{kj}^{(d_2)}\}$
- Let's define $A_{ij}^{(d)} = \max\{f : (d', f) \in A_{ij}, d' \le d\}$.



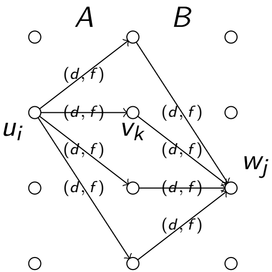
1. Exact product for small distances

- Given d, what's the largest f s.t. $C_{ij}(f) \le d$?
  - $\max_{d_1+d_2=d}(A^{(d_1)} \circledvee B^{(d_2)})_{ij}$
  - Recall that $\circledvee$ is max-min product.
- Let's define $A_{ij}^{(d)} = \max\{f : (d', f) \in A_{ij}, d' \le d\}$.



1. Exact product for small distances

- Given d, what's the largest f s.t. $C_{ij}(f) \le d$?
  - $\max_{d_1+d_2=d}(A^{(d_1)} \otimes B^{(d_2)})_{ij}$
  - Recall that $\otimes$ is max-min product.
- Let's define $A_{ij}^{(d)} = \max\{f : (d', f) \in A_{ij}, d' \le d\}$.
- $O(R^2)$ max-min products suffice to compute the df-matrix C!
- $C_{ij} = \{(\max_{d_1+d_2=d}(A^{(d_1)} \otimes B^{(d_2)})_{ij}, f) : 1 \le d \le 2R\}$



2. Approximate product for arbitrary distances

- Given two df-matrices A, B, compute $C \approx A \star B$.
- Now d can be large($d \leq M$), but we only want C to be approximately correct.

2. Approximate product for arbitrary distances

- Given two df-matrices A, B, compute $C \approx A \star B$.
- Now d can be large($d \le M$), but we only want C to be approximately correct.
- Approximation guarantee: $\forall f, i, j$,
  $(A(f) \star B(f))_{ij} \le C(f)_{ij} \le (1 + \frac{4}{R})(A(f) \star B(f))_{ij}$.

2. Approximate product for arbitrary distances

- Given two df-matrices A, B, compute $C \approx A \star B$.
- Now d can be large($d \leq M$), but we only want C to be approximately correct.
- Approximation guarantee: $\forall f, i, j$, $(A(f) \star B(f))_{ij} \leq C(f)_{ij} \leq (1 + \frac{4}{R})(A(f) \star B(f))_{ij}$.
- time complexity: $O(n^{\frac{3+\omega}{2}} R^2 \log M)$.
  - $O(\log M)$ exact products in which $d \leq R$.

A lemma

Lemma ([Zwick, 1998])

Let A, B be two matrices with entries in $\{0, 1, \ldots, M, +\infty\}$, $C = A \star B$.

A lemma

Lemma ([Zwick, 1998])

Let A, B be two matrices with entries in $\{0, 1, \ldots, M, +\infty\}$, $C = A \star B$. Let R be a power of 2, $\textsc{Scale}(A, M, R)$ be a matrix A' such that

$$A'_{ij} = \begin{cases} \lceil RA_{ij}/M \rceil & \text{if } 0 \le A_{ij} \le M \\ +\infty & \text{otherwise} \end{cases}.$$

A lemma

Lemma ([Zwick, 1998])

Let A, B be two matrices with entries in $\{0, 1, \ldots, M, +\infty\}$, $C = A \star B$. Let R be a power of 2, $\textsc{Scale}(A, M, R)$ be a matrix A' such that

$$A'_{ij} = \begin{cases} \lceil RA_{ij}/M \rceil & \text{if } 0 \leq A_{ij} \leq M \\ +\infty & \text{otherwise} \end{cases}.$$

Define C' as:

$$C' = \min_{\lfloor \log_2 R \rfloor \leq r \leq \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A, 2^r, R) \star \textsc{Scale}(B, 2^r, R))\},$$

A lemma

Lemma ([Zwick, 1998])

Let A, B be two matrices with entries in $\{0, 1, \ldots, M, +\infty\}$, $C = A \star B$. Let R be a power of 2, $\textsc{Scale}(A, M, R)$ be a matrix A' such that

$$A'_{ij} = \begin{cases} \lceil RA_{ij}/M \rceil & \text{if } 0 \le A_{ij} \le M \\ +\infty & \text{otherwise} \end{cases}.$$

Define C' as:

$$C' = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A, 2^r, R) \star \textsc{Scale}(B, 2^r, R))\},$$

then for any i, j, $C_{ij} \le C'_{ij} \le (1 + \frac{4}{R})C_{ij}$.

2. Approximate product for arbitrary distances

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A(f), 2^r, R) \star \textsc{Scale}(B(f), 2^r, R))\}$$

2. Approximate product for arbitrary distances

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A(f), 2^r, R) \star \textsc{Scale}(B(f), 2^r, R))\}$$

- Define $\textsc{Scale}(A, M, R)_{ij} = \{(\lfloor R \cdot d/M \rfloor, f) : (d, f) \in A_{ij}\}$.
  - Scale down every d.

2. Approximate product for arbitrary distances

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A(f), 2^r, R) \star \textsc{Scale}(B(f), 2^r, R))\}$$

- Define $\textsc{Scale}(A, M, R)_{ij} = \{(\lfloor R \cdot d/M \rfloor, f) : (d, f) \in A_{ij}\}$.
  - Scale down every d.
  - $(\textsc{Scale}(A, M, R))(f) = \textsc{Scale}(A(f), M, R)$.

2. Approximate product for arbitrary distances

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A(f), 2^r, R) \star \textsc{Scale}(B(f), 2^r, R))\}$$

- Define $\textsc{Scale}(A, M, R)_{ij} = \{(\lfloor R \cdot d/M \rfloor, f) : (d, f) \in A_{ij}\}$.
  - Scale down every d.
  - $(\textsc{Scale}(A, M, R))(f) = \textsc{Scale}(A(f), M, R)$.

$$C_{ij} = \bigcup_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{((2^r/R) \cdot d, f) : (d, f) \in (\textsc{Scale}(A, 2^r, R) \star \textsc{Scale}(B, 2^r, R))_{ij}\}$$

2. Approximate product for arbitrary distances

$$C(f) = \min_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{(2^r/R) \cdot (\textsc{Scale}(A(f), 2^r, R) \star \textsc{Scale}(B(f), 2^r, R))\}$$

- Define $\textsc{Scale}(A, M, R)_{ij} = \{(\lfloor R \cdot d/M \rfloor, f) : (d, f) \in A_{ij}\}$.
  - Scale down every d.
  - $(\textsc{Scale}(A, M, R))(f) = \textsc{Scale}(A(f), M, R)$.

$$C_{ij} = \bigcup_{\lfloor \log_2 R \rfloor \le r \le \lceil \log_2 M \rceil} \{((2^r/R) \cdot d, f) : (d, f) \in (\textsc{Scale}(A, 2^r, R) \star \textsc{Scale}(B, 2^r, R))_{ij}\}$$

- (This $\star$ is the previous exact product.)

3. Main procedure

- We're ready to approximate the APSP-AF problem.

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f):$ there is an edge with length d and capacity f between i and $j\}$.

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f) :$ there is an edge with length d and capacity f between i and $j\}$.
- Given the approximation algorithm, our main procedure is very straightforward:

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f) :$ there is an edge with length d and capacity f between i and $j\}$.
- Given the approximation algorithm, our main procedure is very straightforward: just take $\log n$ matrix powers!

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f) :$ there is an edge with length d and capacity f between i and $j\}$.
- Given the approximation algorithm, our main procedure is very straightforward: just take $\log n$ matrix powers!
- Let $\tilde{D}^{(0)} = G$, $\tilde{D}^{(i)}$ be an approximation of $\tilde{D}^{(i-1)} \star \tilde{D}^{(i-1)}$.

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f) :$ there is an edge with length d and capacity f between i and $j\}$.
- Given the approximation algorithm, our main procedure is very straightforward: just take $\log n$ matrix powers!
- Let $\tilde{D}^{(0)} = G$, $\tilde{D}^{(i)}$ be an approximation of $\tilde{D}^{(i-1)} \star \tilde{D}^{(i-1)}$.
- It can be proved by induction that $\tilde{D}^{(\lceil \log_2 n \rceil)}$ is an $(1 + \frac{4}{R})^{\lceil \log_2 n \rceil}$-approximation of APSP-AF distances.

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f) :$ there is an edge with length d and capacity f between i and $j\}$.
- Given the approximation algorithm, our main procedure is very straightforward: just take $\log n$ matrix powers!
- Let $\tilde{D}^{(0)} = G$, $\tilde{D}^{(i)}$ be an approximation of $\tilde{D}^{(i-1)} \star \tilde{D}^{(i-1)}$.
- It can be proved by induction that $\tilde{D}^{(\lceil \log_2 n \rceil)}$ is an $(1 + \frac{4}{R})^{\lceil \log_2 n \rceil}$-approximation of APSP-AF distances.
- We set R the smallest power of 2 greater than $4\lceil \log_2 n \rceil / \ln(1 + \epsilon)$, and we're done.

3. Main procedure

- We're ready to approximate the APSP-AF problem.
- Recall that we're given a graph G where $G_{ij} = \{(d, f) :$ there is an edge with length d and capacity f between i and $j\}$.
- Given the approximation algorithm, our main procedure is very straightforward: just take $\log n$ matrix powers!
- Let $\tilde{D}^{(0)} = G$, $\tilde{D}^{(i)}$ be an approximation of $\tilde{D}^{(i-1)} \star \tilde{D}^{(i-1)}$.
- It can be proved by induction that $\tilde{D}^{(\lceil \log_2 n \rceil)}$ is an $(1 + \frac{4}{R})^{\lceil \log_2 n \rceil}$-approximation of APSP-AF distances.
- We set R the smallest power of 2 greater than $4\lceil \log_2 n \rceil / \ln(1 + \epsilon)$, and we're done.
- Time complexity: $O(n^{\frac{3+\omega}{2}} R^2 \log M \log n) = \tilde{O}(n^{\frac{3+\omega}{2}} \epsilon^{-2} \log M)$.

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ:

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work
  - Let $t = n^{\frac{3-\omega}{2}}$, then $O(tn^{\omega} + n^3/t) = O(n^{\frac{3+\omega}{2}})$

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work
  - Let $t = n^{\frac{3-\omega}{2}}$, then $O(tn^{\omega} + n^3/t) = O(n^{\frac{3+\omega}{2}})$
- exact product of df-matrices: $O(tR^2)$ MMs & $O(R^2 n^3/t)$ extra work

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work
  - Let $t = n^{\frac{3-\omega}{2}}$, then $O(tn^{\omega} + n^3/t) = O(n^{\frac{3+\omega}{2}})$
- exact product of *df*-matrices: $O(tR^2)$ MMs & $O(R^2n^3/t)$ extra work
  - It turns out that these $O(tR^2)$ MMs are expressible in $O(t)$ **distance products** of matrices whose elements are $\leq R$.

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work
  - Let $t = n^{\frac{3-\omega}{2}}$, then $O(tn^{\omega} + n^3/t) = O(n^{\frac{3+\omega}{2}})$
- exact product of *df*-matrices: $O(tR^2)$ MMs & $O(R^2n^3/t)$ extra work
  - It turns out that these $O(tR^2)$ MMs are expressible in $O(t)$ **distance products** of matrices whose elements are $\leq R$.
  - Such a distance product can be computed in $\tilde{O}(Rn^{\omega})$ [Zwick, 1998].

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work
  - Let $t = n^{\frac{3-\omega}{2}}$, then $O(tn^{\omega} + n^3/t) = O(n^{\frac{3+\omega}{2}})$
- exact product of *df*-matrices: $O(tR^2)$ MMs & $O(R^2 n^3/t)$ extra work
  - It turns out that these $O(tR^2)$ MMs are expressible in $O(t)$ **distance products** of matrices whose elements are $\leq R$.
  - Such a distance product can be computed in $\tilde{O}(Rn^{\omega})$ [Zwick, 1998].
  - Speedup: $O(tR^2n^{\omega}) \Rightarrow \tilde{O}(tRn^{\omega})$.

(Sketch of) a faster algorithm

- We can reduce the dependency on ϵ: there is an $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$-time algorithm.
- It suffices to compute the exact product in $\tilde{O}(n^{\frac{3+\omega}{2}}R^{3/2})$ time.
- Max-min product [Duan and Pettie, 2009]: $O(t)$ matrix-multiplications & $O(n^3/t)$ extra work
  - Let $t = n^{\frac{3-\omega}{2}}$, then $O(tn^\omega + n^3/t) = O(n^{\frac{3+\omega}{2}})$
- exact product of df-matrices: $O(tR^2)$ MMs & $O(R^2n^3/t)$ extra work
  - It turns out that these $O(tR^2)$ MMs are expressible in $O(t)$ **distance products** of matrices whose elements are $\leq R$.
  - Such a distance product can be computed in $\tilde{O}(Rn^\omega)$ [Zwick, 1998].
  - Speedup: $O(tR^2n^\omega) \Rightarrow \tilde{O}(tRn^\omega)$.
  - Let $t = n^{\frac{3-\omega}{2}}R^{1/2}$ and we're done.

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.
  - Main ingredient: faster max-min product [Duan and Pettie, 2009];

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.
  - Main ingredient: faster max-min product [Duan and Pettie, 2009]; distance-product approximation [Zwick, 1998].

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.
  - Main ingredient: faster max-min product [Duan and Pettie, 2009]; distance-product approximation [Zwick, 1998].
  - We furthermore reduced the time dependency on ϵ: $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W)$

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.
  - Main ingredient: faster max-min product [Duan and Pettie, 2009]; distance-product approximation [Zwick, 1998].
  - We furthermore reduced the time dependency on ϵ: $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W) \Rightarrow \tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$.

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.
  - Main ingredient: faster max-min product [Duan and Pettie, 2009]; distance-product approximation [Zwick, 1998].
  - We furthermore reduced the time dependency on ϵ: $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W) \Rightarrow \tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$.
- Open problems
  - Reducing dependency on n(*i.e.* $\frac{3+\omega}{2}$) requires faster max-min product.

Conclusions and open problems

- Conclusions
  - We present an algorithm for $(1+\epsilon)$-approximating APSP-AF problem in truly-subcubic time.
  - Main ingredient: faster max-min product [Duan and Pettie, 2009]; distance-product approximation [Zwick, 1998].
  - We furthermore reduced the time dependency on ϵ: $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-2}\log W) \Rightarrow \tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-3/2}\log W)$.
- Open problems
  - Reducing dependency on n(*i.e.* $\frac{3+\omega}{2}$) requires faster max-min product. But can APSP-AF be done in $\tilde{O}(n^{\frac{3+\omega}{2}}\epsilon^{-1}\log W)$?

Thank you!

Questions are welcome!

References I

Ausiello, G., Italiano, G. F., Spaccamela, A. M., and Nanni, U. (1991).
Incremental algorithms for minimal length paths.
Journal of Algorithms, 12(4):615–638.

Baswana, S., Hariharan, R., and Sen, S. (2007).
Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths.
Journal of Algorithms, 62(2):74–92.

Bernstein, A. (2016).
Maintaining shortest paths under deletions in weighted directed graphs.
SIAM Journal on Computing, 45(2):548–574.

References II

Bernstein, A. and Roditty, L. (2011).
Improved dynamic algorithms for maintaining approximate shortest paths under deletions.
In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1355–1365. Society for Industrial and Applied Mathematics.

Bose, P., Maheshwari, A., Narasimhan, G., Smid, M., and Zeh, N. (2004).
Approximating geometric bottleneck shortest paths.
Computational Geometry, 29(3):233–249.

Coppersmith, D. and Winograd, S. (1990).
Matrix multiplication via arithmetic progressions.
Journal of Symbolic Computation, 9(3):251–280.

References III

Duan, R. and Pettie, S. (2008).
Bounded-leg distance and reachability oracles.
In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 436–445. Society for Industrial and Applied Mathematics.

Duan, R. and Pettie, S. (2009).
Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths.
In *Twentieth Acm-Siam Symposium on Discrete Algorithms, SODA 2009, New York, Ny, Usa, January*, pages 384–391.

References IV

Henzinger, M., Krinninger, S., Nanongkai, D., and Saranurak, T. (2015).
Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture.
In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 21–30. ACM.

Roditty, L. and Segal, M. (2007).
On bounded leg shortest paths problems.
In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 775–784. Society for Industrial and Applied Mathematics.

References V

Roditty, L. and Zwick, U. (2004).
Dynamic approximate all-pairs shortest paths in undirected graphs.
In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 499–508. IEEE.

Shinn, T.-W. and Takaoka, T. (2013).
Efficient graph algorithms for network analysis.
In *First International Conference on Resource Efficiency in Interorganizational Networks-ResEff 2013*, page 236.

Shinn, T.-W. and Takaoka, T. (2014a).
Combining all pairs shortest paths and all pairs bottleneck paths problems.
In *Latin American Symposium on Theoretical Informatics*, pages 226–237. Springer.

References VI

Shinn, T.-W. and Takaoka, T. (2014b).
Combining the shortest paths and the bottleneck paths problems.
In *Proceedings of the Thirty-Seventh Australasian Computer Science Conference-Volume 147*, pages 13–18. Australian Computer Society, Inc.

Shinn, T.-W. and Takaoka, T. (2015).
Variations on the bottleneck paths problem.
Theoretical Computer Science, 575:10 – 16.
Special Issue on Algorithms and Computation.

Zwick, U. (1998).
All pairs shortest paths in weighted directed graphs – exact and almost exact algorithms.
In *Foundations of Computer Science, 1998. Proceedings. Symposium on*, pages 310–319.