Finding a Maximum Weight Triangle in $O(n^{3-\delta})$ Time, With Applications

Virginia Vassilevska and Ryan Williams
Carnegie Mellon University
The Problem

Input: Graph with real-number weights on the nodes

Task: Find a triangle of maximum weight sum
The Problem

**Input:** Graph with real-number weights on the nodes

**Task:** Find a triangle of maximum weight sum
The Problem

**Input:** Graph with real-number weights on the nodes

**Task:** Find a triangle of maximum weight sum
Past Work
Past Work

[Itai and Rodeh, ’78]

Triangle Detection is in $O(n^{\omega})$ time:

Check if $(A \land (A \times A)) \neq 0$
Past Work

[Itai and Rodeh, ’78]

Triangle Detection is in $O(n^\omega)$ time:

\[
\text{Check if } (A \wedge (A \times A)) \neq 0
\]

Their paper ends with:

“A related problem is finding a minimum weighted circuit in a weighted graph. It is unclear to us whether our methods can be modified to answer this problem too.”
Folklore Result
Folklore Result

**Def.** The *distance product* of $A$ and $B$ is the matrix

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

Def. The **distance product** of $A$ and $B$ is the matrix

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

Observation: **Distance Product can solve Max Weight Triangle**
Folklore Result

**Def.** The **distance product** of \( A \) and \( B \) is the matrix

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

Observation: **Distance Product can solve Max Weight Triangle**

1. Push weights from nodes to edges: \( w(u, v) = (w(u) + w(v))/2 \)  
   (Reduce Node-Weighted Triangle to Edge-Weighted Triangle)

2. Compute \( \text{MAX}_{i,j}\{((−A) \star (−A))[i, j] − A[i, j]\}\)  
   (Min Weight Triangle: \( \text{MIN}_{i,j}\{(A \star A)[i, j] + A[i, j]\}\)
Easy Weighted Triangle Algorithms
Easy Weighted Triangle Algorithms

• [Zwick, ’02] $O(M \cdot n^\omega)$ distance product algorithm

$\implies$ Max Weight Triangle in $O(M \cdot n^\omega)$ (Pseudopolynomial)
Easy Weighted Triangle Algorithms

- [Zwick, '02] $O(M \cdot n^\omega)$ distance product algorithm
  $\implies$ Max Weight Triangle in $O(M \cdot n^\omega)$ (Pseudopolynomial)

- [Chan, '05] $O(n^3/\log n)$ distance product
  $\implies$ Max Weighted Triangle in $O(n^3/\log n)$
Easy Weighted Triangle Algorithms

• [Zwick, ’02] \( O(M \cdot n^\omega) \) distance product algorithm
  \( \Rightarrow \) Max Weight Triangle in \( O(M \cdot n^\omega) \) (Pseudopolynomial)

• [Chan, ’05] \( O(n^3 / \log n) \) distance product
  \( \Rightarrow \) Max Weighted Triangle in \( O(n^3 / \log n) \)

Truly Sub-Cubic Algorithm?
Talk Outline

- Deterministic Algorithm
  \[ O(B \cdot n^{(3+\omega)/2}) \leq O(B \cdot n^{2.688}) \], where \( B \) is the bit precision

- Randomized (Strongly Polynomial) Algorithm
  \[ O(n^{(3+\omega)/2 \log n}) \leq O(n^{2.688}) \]

- Some Applications
Deterministic Algorithm

Key Steps:

• Suffices to check if there’s a triangle of weight $\geq K$

• Compute a matrix $A'$ s.t.

$$A'[i, j] = |\{k : i \rightarrow k \rightarrow j, w(i) + w(k) + w(j) \geq K\}|.$$

• Check if $\exists i, j$ where $A[i, j]$ and $A'[i, j]$ are non-zero.
Computing $C$

**Def.** The **dominance product** of $A$ and $B$ is the matrix $C$ s.t.

$$C[i, j] = |\{k : A[i, k] \leq B[k, j]\}|$$
Computing $C$

**Def.** The *dominance product* of $A$ and $B$ is the matrix $C$ s.t.

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

**Thm.** (Matousek) Dominance Product can be computed in $n^{(3+\omega)/2}$ time. (Sketched in next few slides)
Computing $C$

**Def.** The **dominance product** of $A$ and $B$ is the matrix $C$ s.t.

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

**Thm.** (Matousek) Dominance Product can be computed in $n^{(3+\omega)/2}$ time. (Sketched in next few slides)

**Thm.** If Dominance Product is in $O(f(n))$ time, then can check if there’s a triangle of weight $\geq K$, in $O(f(n) + n^2)$ time. (Virginia will prove this)
Dominance Product in $\mathcal{H}^{(3+\omega)/2}$

\[ (C[i, j] = |\{k : A[i, k] \leq B[k, j]\}|) \]
**Dominance Product in** $\mathcal{N}^{(3+\omega)/2}$

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

**Idea 1:** Just care about the sorted order of coordinates

$\implies$ WLOG each column of $A$ and each row of $B$ is a permutation of $[n]$. 
Dominance Product in $n^{(3+\omega)/2}$

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

**Idea 1:** Just care about the sorted order of coordinates

$\implies$ WLOG each column of $A$ and each row of $B$ is a permutation of $[n]$.

Make $n$ sorted lists $L_1, \ldots, L_n$, where

$L_i$ has the $i$th column of $A$ and the $i$th row of $B$
Dominance Product in \( n^{(3+\omega)/2} \)

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

**Idea 1:** Just care about the sorted order of coordinates

\( \implies \) WLOG each column of \( A \) and each row of \( B \) is a permutation of \([n]\).

Make \( n \) sorted lists \( L_1, \ldots, L_n \), where

\( L_i \) has the \( i \)th column of \( A \) and the \( i \)th row of \( B \)

Partition each \( L_i \) into “buckets” with \( s \) elements in each bucket (roughly \( 2n/s \) buckets in total)

\[
\begin{array}{cccccc}
L_k & 2n \\
\hline
\ldots & A[1, k] & \ldots & A[2, k] & \ldots & B[k, 1] & \ldots & B[k, 2] & \ldots \\
\hline
s & s & s & s & s \\
\end{array}
\]
Dominance Product in $n^{(3+\omega)/2}$, Cont.

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

Idea 2: Two types of data are counted in $C'$:
Dominance Product in $n^{(3+\omega)/2}$, Cont.

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

**Idea 2:** Two types of data are counted in $C$:

1. Pairs $(A[i, k], B[k, j])$ such that $A[i, k] \leq B[k, j]$, but $A[i, k]$ and $B[k, j]$ fall in the same bucket of $L_k$
   - Only $O(n^2s)$ possible pairs of this form
   - Can compute these in $O(1)$ amortized time
Dominance Product in $n^{(3+\omega)/2}$, Cont.

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

Idea 2: Two types of data are counted in $C$:

2. Pairs $(A[i, k], B[k, j])$ such that $A[i, k] \leq B[k, j]$, but $A[i, k]$ and $B[k, j]$ fall in different buckets of $L_k$. 

• Can count these using $2n/s$ matrix multiplications
  (One matrix multiply for each bucket)
Deterministic Algorithm: Outline
Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least $K$? → dominance product instance.
Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least $K$? \(\rightarrow\) dominance product instance.

2. Do binary search on $K$ to find the maximum weight $W$ of a triangle.
Deterministic Algorithm: Outline

1. Does there exist a triangle of weight sum at least $K$? → dominance product instance.

2. Do binary search on $K$ to find the maximum weight $W$ of a triangle.

3. Find a triangle of weight $W$. 
Step 1: Given $K$, reduce to dominance product instance.

Vertex $i \in V \rightarrow$
Step 1: Given $K$, reduce to dominance product instance.

Vertex $i \in V \rightarrow$

- row vector $A[i, ;] = (A[i, 1], \ldots, A[i, n])$ s.t.

$$A[i, j] = \begin{cases} 
K - w(i) & \text{if there is an edge from } i \text{ to } j \\
\infty & \text{otherwise.}
\end{cases}$$
Step 1: Given $K$, reduce to dominance product instance.

Vertex $i \in V \rightarrow$

- row vector $A[i, ;] = (A[i, 1], \ldots, A[i, n])$ s.t.
  \[
  A[i, j] = \begin{cases} 
  K - w(i) & \text{if there is an edge from } i \text{ to } j \\
  \infty & \text{otherwise}.
  \end{cases}
  \]

- column vector $B[; , i] = (B[1, i], \ldots, B[n, i])$ s.t.
  \[
  B[j, i] = \begin{cases} 
  w(i) + w(j) & \text{if there is an edge from } i \text{ to } j \\
  -\infty & \text{otherwise}.
  \end{cases}
  \]
Step 1: Given $K$, reduce to dominance product instance.

Vertex $i \in V \rightarrow$

- row vector $A[i, ;] = (A[i, 1], \ldots, A[i, n])$ s.t.
  
  $$A[i, j] = \begin{cases} K - w(i) & \text{if there is an edge from } i \text{ to } j \\ \infty & \text{otherwise.} \end{cases}$$

- column vector $B[; , i] = (B[1, i], \ldots, B[n, i])$ s.t.

  $$B[j, i] = \begin{cases} w(i) + w(j) & \text{if there is an edge from } i \text{ to } j \\ -\infty & \text{otherwise.} \end{cases}$$

$$A[i, j] \leq B[j, k] \iff K \leq w(i) + w(k) + w(j) \text{ and } (i, j), (j, k) \in E$$
Runtime
Runtime

Let $B$ be the max number of bits needed to represent a weight.
Runtime

Let $B$ be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$. 
Runtime

Let $B$ be the max number of bits needed to represent a weight.

Then the binary search calls at most $O(B)$ dominance computations, and hence the runtime is $O(B \cdot n^{\frac{3+\omega}{2}})$.

But this algorithm is NOT strongly polynomial because of the binary search.
A Strongly Polynomial Randomized Algorithm: Outline

1. show how to sample a triangle of weight in any interval $[W_1, W_2]$ efficiently uniformly at random

2. search using the weights of triangles chosen at random
Getting a uniform random triangle in $[W_1, W_2]$
Getting a uniform random triangle in $[W_1, W_2]$ 

A dominance computation gives us the number of coordinates for which a vector dominates another.
Getting a uniform random triangle in $[W_1, W_2]$  

A dominance computation gives us the number of coordinates for which a vector dominates another.

Hence for each $(i, j)$ we can get
Getting a uniform random triangle in \([W_1, W_2]\)

A dominance computation gives us the number of coordinates for which a vector dominates another.

Hence for each \((i, j)\) we can get

- the number \(E_{ij}^1\) of \(k\) such that

\[
\ldots, w(i) + w(k), \ldots \text{ dominated in coord. } k \text{ by } \ldots, W_2 - w(j), \ldots
\]

\(i \rightarrow k \rightarrow j\) and \(w(i) + w(j) + w(k) \leq W_2\)
Getting a uniform random triangle in $[W_1, W_2]$.

A dominance computation gives us the number of coordinates for which a vector dominates another.

Hence for each $(i, j)$ we can get

- the number $E_{ij}^1$ of $k$ such that $(\ldots, w(i) + w(k), \ldots)$ dominated in coord. $k$ by $(\ldots, W_2 - w(j), \ldots)$
  \[i \rightarrow k \rightarrow j \text{ and } w(i) + w(j) + w(k) \leq W_2\]

- the number $E_{ij}^2$ of $k$ such that $i \rightarrow k \rightarrow j$ and $w(i) + w(j) + w(k) < W_1$
Getting a uniform random triangle in $[W_1, W_2]$

A dominance computation gives us the number of coordinates for which a vector dominates another.

Hence for each $(i, j)$ we can get

- the number $E_{ij}^1$ of $k$ such that
  $$(\ldots, w(i) + w(k), \ldots)$$
  dominated in coord. $k$ by $(\ldots, W_2 - w(j), \ldots)$
  $i \rightarrow k \rightarrow j$ and $w(i) + w(j) + w(k) \leq W_2$

- the number $E_{ij}^2$ of $k$ such that
  $i \rightarrow k \rightarrow j$ and $w(i) + w(j) + w(k) < W_1$

- the number $E_{ij} = (E_{ij}^1 - E_{ij}^2)$ of $k$ such that
  $i \rightarrow k \rightarrow j$ and $W_1 \leq w(i) + w(j) + w(k) \leq W_2$
Getting a uniform random triangle in $[W_1, W_2]$

Recall: $E_{ij}$ is the number of $k$ such that $i \rightarrow k \rightarrow j$ and $W_1 \leq w(i) + w(j) + w(k) \leq W_2$
Getting a uniform random triangle in $[W_1, W_2]$

Recall: $E_{ij}$ is the number of $k$ such that $i \rightarrow k \rightarrow j$ and $W_1 \leq w(i) + w(j) + w(k) \leq W_2$

Uniformly sample an edge from a triangle in $[W_1, W_2]$:

$f = \sum_{(i,j) \in E} E_{ij}$ is $3 \times [\text{number of triangles in } [W_1, W_2] \text{]}$. 
Getting a uniform random triangle in $[W_1, W_2]$

Recall: $E_{ij}$ is the number of $k$ such that $i \rightarrow k \rightarrow j$ and $W_1 \leq w(i) + w(j) + w(k) \leq W_2$

Uniformly sample an edge from a triangle in $[W_1, W_2]$: $f = \sum_{(i,j) \in E} E_{ij}$ is $3 \times \text{[number of triangles in } [W_1, W_2] \text{]}$. Pick each $(i, j) \in E$ with probability $E_{ij}/f$. 
Getting a uniform random triangle in \([W_1, W_2]\)

Recall: \(E_{ij}\) is the number of \(k\) such that \(i \to k \to j\) and 
\[W_1 \leq w(i) + w(j) + w(k) \leq W_2\]

Uniformly sample an edge from a triangle in \([W_1, W_2]\):
\[f = \sum_{(i,j) \in E} E_{ij}\] is \(3 \times [\text{number of triangles in } [W_1, W_2]]\).

Pick each \((i, j) \in E\) with probability \(E_{ij}/f\).

Uniformly sample a triangle in \([W_1, W_2]\): Let \(S_{ij}\) be the common neighbors of \(i\) and \(j\).
Getting a uniform random triangle in $[W_1, W_2]$

Recall: $E_{ij}$ is the number of $k$ such that $i \rightarrow k \rightarrow j$ and $W_1 \leq w(i) + w(j) + w(k) \leq W_2$

Uniformly sample an edge from a triangle in $[W_1, W_2]$:

$f = \sum_{(i,j) \in E} E_{ij}$ is $3 \times \lfloor \text{number of triangles in } [W_1, W_2] \rfloor$.

Pick each $(i, j) \in E$ with probability $E_{ij}/f$.

Uniformly sample a triangle in $[W_1, W_2]$: Let $S_{ij}$ be the common neighbors of $i$ and $j$.

Pick $k \in S_{ij}$ uniformly at random.
Getting a uniform random triangle in $[W_1, W_2]$

Recall: $E_{ij}$ is the number of $k$ such that $i \to k \to j$ and $W_1 \leq w(i) + w(j) + w(k) \leq W_2$

Uniformly sample an edge from a triangle in $[W_1, W_2]$:

$f = \sum_{(i,j) \in E} E_{ij}$ is $3 \times [\text{number of triangles in } [W_1, W_2]]$.

Pick each $(i, j) \in E$ with probability $E_{ij}/f$.

Uniformly sample a triangle in $[W_1, W_2]$: Let $S_{ij}$ be the common neighbors of $i$ and $j$.

Pick $k \in S_{ij}$ uniformly at random.

$\{i, j, k\}$ is a random triangle with weight in $[W_1, W_2]$. 
Strongly Polynomial Algorithm

1. Let $M = 3 \cdot \max_{i \in V} w(i)$ and $K = 0$.


3. Check if there exists a triangle of weight $> K$. If not, return $T$.

4. Repeat from 2.
Strongly Polynomial Algorithm

1. Let $M = 3 \cdot \max_{i \in V} w(i)$ and $K = 0$.


3. Check if there exists a triangle of weight $> K$. If not, return $T$.

4. Repeat from 2.

The algorithm will terminate in $O(n^{3+\omega \over 2} \log n)$ expected worst case time.
Applications
Applications

• maximum node-weighted \(3K\)-clique in \(\tilde{O}(n^{\frac{3+\omega}{2}K})\)
Applications

• maximum node-weighted $3K$-clique in $\tilde{O}(n^{\frac{3+\omega}{2}}K)$

• for any $3K$-node graph $H$, maximum node-weighted $H$-subgraph in $\tilde{O}(n^{\frac{3+\omega}{2}}K)$
Applications

• maximum node-weighted $3K$-clique in $\tilde{O}(n^{\frac{3+\omega}{2}K})$

• for any $3K$-node graph $H$, maximum node-weighted $H$-subgraph in $\tilde{O}(n^{\frac{3+\omega}{2}K})$

• generalized $K$-SUM
Applications

• maximum node-weighted $3K$-clique in $\tilde{O}(n^{\frac{3+\omega}{2}}K)$

• for any $3K$-node graph $H$, maximum node-weighted $H$-subgraph in $\tilde{O}(n^{\frac{3+\omega}{2}}K)$

• generalized $K$-SUM

• computing $K$ most significant bits of distance product in $O(2^K \cdot n^{\frac{3+\omega}{2}} \log W \log n)$ ...
Conclusions
Conclusions

- [VWY] have recently discovered a faster algorithm for node-weighted triangle: $O(n^{2.58})$ using rectangular matrix multiplication (ICALP 2006)
Conclusions

• [VWY] have recently discovered a faster algorithm for node-weighted triangle: $O(n^{2.58})$ using rectangular matrix multiplication (ICALP 2006)

• Can we use our approach to edge-weighted triangle? Important stepping stone towards truly sub-cubic APSP?
Conclusions

• [VWY] have recently discovered a faster algorithm for node-weighted triangle: $O(n^{2.58})$ using rectangular matrix multiplication (ICALP 2006)

• Can we use our approach to edge-weighted triangle? Important stepping stone towards truly sub-cubic APSP?

• Conjecture:
  Dominance product can be computed in $O(n^{\omega+o(1)})$ time.
Thank You!