Near-optimal Algorithms for Shortest Paths in Weighted Unit-Disk Graphs

Haitao Wang[1] Jie Xue[2]

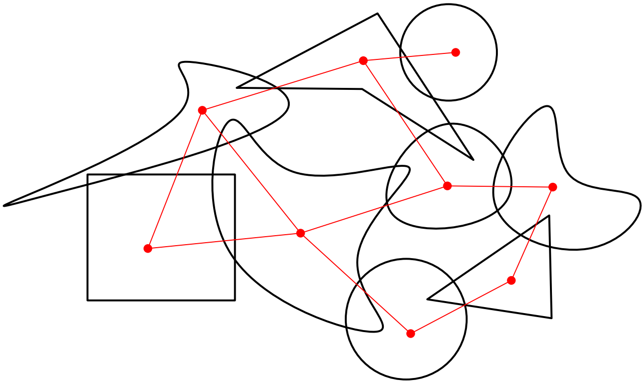
[1]Utah State University

[2]University of Minnesota, Twin Cities

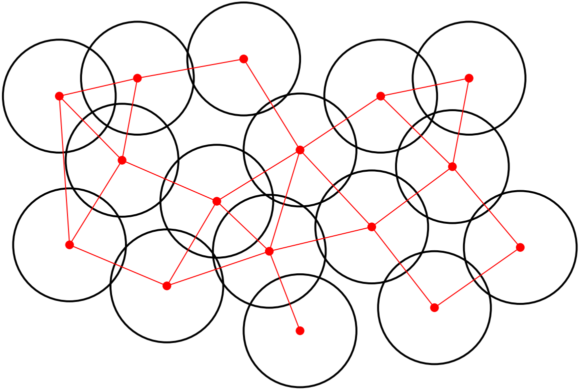
Student Presentation

Background

- **Geometric intersection graphs**
  The intersection graph of a set of geometric objects



- **Unit-disk graphs (UDGs)**
  The intersection graph of unit-disks or disks of identical radii



Background

- **Single-source shortest path (SSSP) problem**
  Given a positively weighted graph $G = (V, E, w)$ and a source $s \in V$, compute the shortest paths from s to all the other vertices of G.

- **Single-source shortest path (SSSP) problem**
  Given a positively weighted graph $G = (V, E, w)$ and a source $s \in V$, compute the shortest paths from s to all the other vertices of G.

- SSSP on UDGs?

Background

- **Single-source shortest path (SSSP) problem**
  Given a positively weighted graph $G = (V, E, w)$ and a source $s \in V$, compute the shortest paths from s to all the other vertices of G.
- SSSP on UDGs?
- **Two ways to weight a UDG**
  1. Edges are weighted identically (Unweighted UDGs)
  2. Edges are weighted using Euclidean distances between the disk centers (Weighted UDGs)

Background

- **Classical SSSP algorithms**
  - Dijkstra's algorithm
  - Johnson's algorithm
  - Bellman-Ford algorithm
  - ...

Background

- **Classical SSSP algorithms**
  - Dijkstra's algorithm
  - Johnson's algorithm
  - Bellman-Ford algorithm
  - ...
- These algorithms requires $\Omega(|E|)$ time for solving SSSP.
- It is a lower bound for general graphs, due to the $\Omega(|E|)$ input size.

Background

- **Classical SSSP algorithms**
  - Dijkstra's algorithm
  - Johnson's algorithm
  - Bellman-Ford algorithm
  - ...
- These algorithms requires $\Omega(|E|)$ time for solving SSSP.
- It is a lower bound for general graphs, due to the $\Omega(|E|)$ input size.
- The input size for an n-vertex UDG is only $O(n)$.
- However, $|E| = \Omega(n^2)$ in an n-vertex UDG in worst case.

Background

- **Classical SSSP algorithms**
  - Dijkstra's algorithm
  - Johnson's algorithm
  - Bellman-Ford algorithm
  - ...
- These algorithms requires $\Omega(|E|)$ time for solving SSSP.
- It is a lower bound for general graphs, due to the $\Omega(|E|)$ input size.
- The input size for an n-vertex UDG is only $O(n)$.
- However, $|E| = \Omega(n^2)$ in an n-vertex UDG in worst case.
- Maybe we can break the $\Omega(|E|)$ lower bound for UDGs?

Background

- **SSSP on unweighted UDGs**
  - $O(n \log n)$ time and $O(n)$ space by [Cabello and Jejčič 2015]
  - $O(n)$ time and $O(n)$ space after presorting by [Chan and Skrepetos 2016]

Background

- **SSSP on unweighted UDGs**
  - $O(n \log n)$ time and $O(n)$ space by [Cabello and Jejčič 2015]
  - $O(n)$ time and $O(n)$ space after presorting by [Chan and Skrepetos 2016]
- **SSSP on weighted UDGs**
  - $O(n^{1+\delta})$ time and $O(n^{1+\delta})$ space for any $\delta > 0$ by [Cabello and Jejčič 2015]
  - $O(n \log^{12+o(1)} n)$ expected time and $O(n \log^3 n)$ space (randomized) by [Kaplan et al. 2017]
  - $O(n \log n/\varepsilon^2)$ time and $O(n/\varepsilon^2)$ space for $(1+\varepsilon)$-approximation by [Chan and Skrepetos 2016]

Our results

- **The subject of this work:** SSSP on weighted UDGs

Our results

- **The subject of this work:** SSSP on weighted UDGs

Theorem (Exact algorithm)

There is an SSSP algorithm on weighted UDGs using $O(n \log^2 n)$ time and $O(n)$ space, where n is the input size.

Our results

- **The subject of this work:** SSSP on weighted UDGs

Theorem (Exact algorithm)

There is an SSSP algorithm on weighted UDGs using $O(n \log^2 n)$ time and $O(n)$ space, where n is the input size.

Theorem (Approximation algorithm)

There is a $(1+\varepsilon)$-approximate SSSP algorithm on weighted UDGs using $O(n \log n + n \log^2(1/\varepsilon))$ time and $O(n)$ space, where n is the input size.

Our results

- Our results are achieved by relating SSSP on weighted UDGs to the offline insertion-only weighted nearest-neighbor (OIWNN) problem.

Our results

- Our results are achieved by relating SSSP on weighted UDGs to the offline insertion-only weighted nearest-neighbor (OIWNN) problem.
- **The OIWNN problem in $\mathbb{R}^2$**
  [Input] a sequence of n operations each of which is one of
  Insert(s) - insert a new weighted site $s \in \mathbb{R}^2$
  Query(q) - query the WNN of $q \in \mathbb{R}^2$ among the current sites

Our results

- Our results are achieved by relating SSSP on weighted UDGs to the offline insertion-only weighted nearest-neighbor (OIWNN) problem.
- **The OIWNN problem in** $\mathbb{R}^2$
  [Input] a sequence of n operations each of which is one of
  Insert(s) - insert a new weighted site $s \in \mathbb{R}^2$
  Query(q) - query the WNN of $q \in \mathbb{R}^2$ among the current sites
  [Goal] answer all queries

Our results

- Our results are achieved by relating SSSP on weighted UDGs to the offline insertion-only weighted nearest-neighbor (OIWNN) problem.
- **The OIWNN problem in $\mathbb{R}^2$**
  [Input] a sequence of n operations each of which is one of
  Insert(s) - insert a new weighted site $s \in \mathbb{R}^2$
  Query(q) - query the WNN of $q \in \mathbb{R}^2$ among the current sites
  [Goal] answer all queries
- We reduce SSSP on weighted UDGs to the OIWNN problem in $\mathbb{R}^2$.

Our results

Theorem (Exact)

If the OIWNN problem with n operations can be solved in $f(n)$ time, then SSSP on weighted UDGs can be solved in $O(n \log n + f(n))$ time.

Our results

Theorem (Exact)

If the OIWNN problem with n operations can be solved in $f(n)$ time, then SSSP on weighted UDGs can be solved in $O(n \log n + f(n))$ time.

- We show that $f(n) = O(n \log^2 n)$ (D&C + WVD).
- This is the bottleneck of our algorithm.

Our results

Theorem (Exact)

If the OIWNN problem with n operations can be solved in $f(n)$ time, then SSSP on weighted UDGs can be solved in $O(n \log n + f(n))$ time.

- We show that $f(n) = O(n \log^2 n)$ (D&C + WVD).
- This is the bottleneck of our algorithm.

Theorem (Approximation)

If the OIWNN problem with n operations in which at most k operations are insertions can be solved in $f(n, k)$ time, then $(1 + \varepsilon)$-approximate SSSP on weighted UDGs can be solved in $O(n \log n + f(n, 1/\varepsilon))$ time.

Our results

Theorem (Exact)

If the OIWNN problem with n operations can be solved in $f(n)$ time, then SSSP on weighted UDGs can be solved in $O(n \log n + f(n))$ time.

- We show that $f(n) = O(n \log^2 n)$ (D&C + WVD).
- This is the bottleneck of our algorithm.

Theorem (Approximation)

If the OIWNN problem with n operations in which at most k operations are insertions can be solved in $f(n, k)$ time, then $(1+\varepsilon)$-approximate SSSP on weighted UDGs can be solved in $O(n \log n + f(n, 1/\varepsilon))$ time.

- We show that $f(n, k) = O(n \log^2 k)$ (D&C + WVD).

The exact SSSP algorithm

- For convenience, assume UDG is defined by disks of radii $\frac{1}{2}$.

The exact SSSP algorithm

- For convenience, assume UDG is defined by disks of radii $\frac{1}{2}$.
- Given n disks of radii $\frac{1}{2}$, let S be the set of the disk centers.

The exact SSSP algorithm

- For convenience, assume UDG is defined by disks of radii $\frac{1}{2}$.
- Given n disks of radii $\frac{1}{2}$, let S be the set of the disk centers.
- Two points $a, b \in S$ are connected by an edge iff $\|a - b\| \leq 1$. (The edge is weighted by $\|a - b\|$.)

The exact SSSP algorithm

- For convenience, assume UDG is defined by disks of radii $\frac{1}{2}$.
- Given n disks of radii $\frac{1}{2}$, let S be the set of the disk centers.
- Two points $a, b \in S$ are connected by an edge iff $\|a - b\| \leq 1$. (The edge is weighted by $\|a - b\|$.)
- Let $s \in S$ be a given source.
- Our goal is to compute a table dist$[\cdot]$, where dist$[a]$ stores the length of the shortest path from s to a, for all $a \in S$.

The exact SSSP algorithm

Dijkstra's algorithm

[Input] $G = (V, E, w)$ and $s \in V$

The exact SSSP algorithm

Dijkstra's algorithm

[Input] $G = (V, E, w)$ and $s \in V$

1. $\text{dist}[s] \leftarrow 0$, $\text{dist}[a] \leftarrow \infty$ for all $a \in V \backslash \{s\}$, $A \leftarrow V$
2. Pick $c \in A$ with the smallest $\text{dist}[c]$
3. For all neighbors $b \in A$ of c, $\text{dist}[b] \leftarrow \min\{\text{dist}[b], \text{dist}[c] + w(b, c)\}$
4. $A \leftarrow A \backslash \{c\}$, go to **Step 2** if $A \neq \emptyset$

The exact SSSP algorithm

Dijkstra's algorithm

[Input] $G = (V, E, w)$ and $s \in V$

1. $\text{dist}[s] \leftarrow 0$, $\text{dist}[a] \leftarrow \infty$ for all $a \in V \backslash \{s\}$, $A \leftarrow V$
2. Pick $c \in A$ with the smallest $\text{dist}[c]$
3. For all neighbors $b \in A$ of c, $\text{dist}[b] \leftarrow \min\{\text{dist}[b], \text{dist}[c] + w(b, c)\}$
4. $A \leftarrow A \backslash \{c\}$, go to **Step 2** if $A \neq \emptyset$

- The previous works [Cabello and Jejčič 2015] and [Kaplan et al. 2017] use Dijkstra's algorithm + dynamic bichromatic closest pair

The exact SSSP algorithm

Dijkstra's algorithm

[Input] $G = (V, E, w)$ and $s \in V$

1. $\text{dist}[s] \leftarrow 0$, $\text{dist}[a] \leftarrow \infty$ for all $a \in V \backslash \{s\}$, $A \leftarrow V$
2. Pick $c \in A$ with the smallest $\text{dist}[c]$
3. For all neighbors $b \in A$ of c, $\text{dist}[b] \leftarrow \min\{\text{dist}[b], \text{dist}[c] + w(b, c)\}$
4. $A \leftarrow A \backslash \{c\}$, go to **Step 2** if $A \neq \emptyset$

- The previous works [Cabello and Jejčič 2015] and [Kaplan et al. 2017] use Dijkstra's algorithm + dynamic bichromatic closest pair
- The framework of our SSSP algorithm is different from Dijkstra's, but it exploits the basic intuition of Dijkstra's.

The exact SSSP algorithm

- Define an operation UPDATE as follows.

The exact SSSP algorithm

- Define an operation UPDATE as follows.
- UPDATE(U, V) for $U, V \subseteq S$
  1. $\text{dist}'[u] \leftarrow \text{dist}[u]$ for all $u \in U$
  2. For each $v \in V$, find the neighbor $u_v \in U$ of v that minimizes $\text{dist}'[u_v] + \|u_v - v\|$
  3. $\text{dist}[v] \leftarrow \min\{\text{dist}[v], \text{dist}'[u_v] + \|u_v - v\|\}$ for all $v \in V$

The exact SSSP algorithm

- Define an operation UPDATE as follows.
- UPDATE(U, V) for $U, V \subseteq S$
  1. $\text{dist}'[u] \leftarrow \text{dist}[u]$ for all $u \in U$
  2. For each $v \in V$, find the neighbor $u_v \in U$ of v that minimizes $\text{dist}'[u_v] + \|u_v - v\|$
  3. $\text{dist}[v] \leftarrow \min\{\text{dist}[v], \text{dist}'[u_v] + \|u_v - v\|\}$ for all $v \in V$
- Roughly speaking, UPDATE(U, V) uses the shortest-path information of U to update the shortest-path information of V.

The exact SSSP algorithm

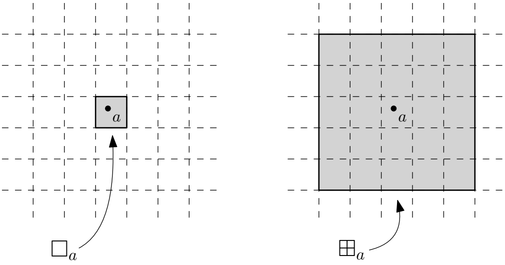
- Define an operation UPDATE as follows.
- UPDATE(U, V) for $U, V \subseteq S$
  1. $\text{dist}'[u] \leftarrow \text{dist}[u]$ for all $u \in U$
  2. For each $v \in V$, find the neighbor $u_v \in U$ of v that minimizes $\text{dist}'[u_v] + \|u_v - v\|$
  3. $\text{dist}[v] \leftarrow \min\{\text{dist}[v], \text{dist}'[u_v] + \|u_v - v\|\}$ for all $v \in V$
- Roughly speaking, UPDATE(U, V) uses the shortest-path information of U to update the shortest-path information of V.
- The table dist$'$ is used for lazy update in case $U \cap V \neq \emptyset$.

The exact SSSP algorithm

- **First step:** build a grid Γ of width $\frac{1}{2}$ on the plane

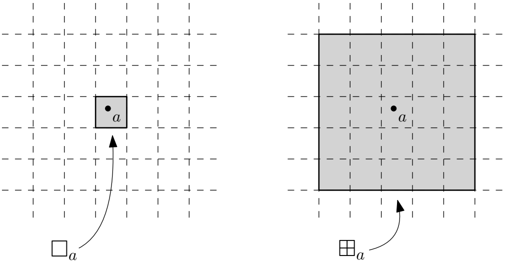
The exact SSSP algorithm

- **First step:** build a grid Γ of width $\frac{1}{2}$ on the plane
- $\square_a$ = the cell of Γ containing a
  $\boxplus_a$ = the 5×5 patch centered at $\square_a$



The exact SSSP algorithm

- **First step:** build a grid Γ of width $\frac{1}{2}$ on the plane
- $\square_a$ = the cell of Γ containing a
  $\boxplus_a$ = the 5×5 patch centered at $\square_a$



- All points in $S \cap \square_a$ are neighbors of a.
- All neighbors of a are in $S \cap \boxplus_a$.

The exact SSSP algorithm

- **Our SSSP algorithm**
  1. dist[s] $\leftarrow 0$, dist[a] $\leftarrow \infty$ for all $a \in V \backslash \{s\}$, $A \leftarrow V$
  2. Pick $c \in A$ with the smallest dist[c]
  3. $\textsc{Update}(A \cap \boxplus_c, A \cap \square_c)$
  4. $\textsc{Update}(A \cap \square_c, A \cap \boxplus_c)$
  5. $A \leftarrow A \backslash \square_c$, go to **Step 2** if $A \neq \emptyset$

The exact SSSP algorithm

- **Our SSSP algorithm**
  1. $\text{dist}[s] \leftarrow 0$, $\text{dist}[a] \leftarrow \infty$ for all $a \in V \backslash \{s\}$, $A \leftarrow V$
  2. Pick $c \in A$ with the smallest $\text{dist}[c]$
  3. $\textsc{Update}(A \cap \boxplus_c, A \cap \square_c)$
  4. $\textsc{Update}(A \cap \square_c, A \cap \boxplus_c)$
  5. $A \leftarrow A \backslash \square_c$, go to **Step 2** if $A \neq \emptyset$
- Convert our algorithm to Dijkstra's algorithm?
  Remove **Step 3** and replace $\square_c$ with $\{c\}$

The exact SSSP algorithm

- **Our SSSP algorithm**
  1. $\text{dist}[s] \leftarrow 0$, $\text{dist}[a] \leftarrow \infty$ for all $a \in V \backslash \{s\}$, $A \leftarrow V$
  2. Pick $c \in A$ with the smallest $\text{dist}[c]$
  3. $\textsc{Update}(A \cap \boxplus_c, A \cap \square_c)$
  4. $\textsc{Update}(A \cap \square_c, A \cap \boxplus_c)$
  5. $A \leftarrow A \backslash \square_c$, go to **Step 2** if $A \neq \emptyset$

- Convert our algorithm to Dijkstra's algorithm?
  Remove **Step 3** and replace $\square_c$ with $\{c\}$

- If we forget **Step 3**, the main difference between Dijkstra's and ours is
  - Dijkstra's considers the single point c in each iteration.
  - Ours considers all points in $\square_c$ in each iteration.

The exact SSSP algorithm

- Why do we need **Step 3**?

The exact SSSP algorithm

- Why do we need **Step 3**?
- In Dijkstra's algorithm, when c is chosen, dist[c] is correct.
- So after using c to update its neighbors, removing c from A is safe.

The exact SSSP algorithm

- Why do we need **Step 3**?
- In Dijkstra's algorithm, when c is chosen, dist[c] is correct.
- So after using c to update its neighbors, removing c from A is safe.
- In our algorithm, when c is chosen, we cannot guarantee that dist[a] is correct for all $a \in A \cap \Box_c$.
- But we need the correctness to do **Step 4** and **Step 5** safely.

The exact SSSP algorithm

- Why do we need **Step 3**?
- In Dijkstra's algorithm, when c is chosen, dist$[c]$ is correct.
- So after using c to update its neighbors, removing c from A is safe.
- In our algorithm, when c is chosen, we cannot guarantee that dist$[a]$ is correct for all $a \in A \cap \square_c$.
- But we need the correctness to do **Step 4** and **Step 5** safely.
- **Step 3** is used to make dist$[a]$ correct for all $a \in A \cap \square_c$.

The exact SSSP algorithm

- Why do we need **Step 3**?

- In Dijkstra's algorithm, when c is chosen, dist[c] is correct.
- So after using c to update its neighbors, removing c from A is safe.

- In our algorithm, when c is chosen, we cannot guarantee that dist[a] is correct for all $a \in A \cap \Box_c$.
- But we need the correctness to do **Step 4** and **Step 5** safely.
- **Step 3** is used to make dist[a] correct for all $a \in A \cap \Box_c$.

Lemma

After **Step 3** *of our algorithm,* dist[a] *equals to the length of the shortest path from s to a for all $a \in A \cap \Box_c$.*

The exact SSSP algorithm

- How to efficiently implement our algorithm?

The exact SSSP algorithm

- How to efficiently implement our algorithm?
- The critical steps: **Step 3** and **Step 4**

The exact SSSP algorithm

- How to efficiently implement our algorithm?
- The critical steps: **Step 3** and **Step 4**
- **Step 3.** $\textsc{Update}(A \cap \boxplus_c, A \cap \square_c)$
  1. $\text{dist}'[u] \leftarrow \text{dist}[u]$ for all $u \in A \cap \boxplus_c$
  2. For each $v \in A \cap \square_c$, find its neighbor $u_v \in A \cap \boxplus_c$ that minimizes $\text{dist}'[u_v] + \|u_v - v\|$
  3. $\text{dist}[v] \leftarrow \min\{\text{dist}[v], \text{dist}'[u_v] + \|u_v - v\|\}$ for all $v \in A \cap \square_c$

The exact SSSP algorithm

- How to efficiently implement our algorithm?
- The critical steps: **Step 3** and **Step 4**
- **Step 3.** $\textsc{Update}(A \cap \boxplus_c, A \cap \square_c)$
  1. $\text{dist}'[u] \leftarrow \text{dist}[u]$ for all $u \in A \cap \boxplus_c$
  2. For each $v \in A \cap \square_c$, find its neighbor $u_v \in A \cap \boxplus_c$ that minimizes $\text{dist}'[u_v] + \|u_v - v\|$
  3. $\text{dist}[v] \leftarrow \min\{\text{dist}[v], \text{dist}'[u_v] + \|u_v - v\|\}$ for all $v \in A \cap \square_c$
- What if we remove the constraint that u_v is a neighbor of v?

The exact SSSP algorithm

- How to efficiently implement our algorithm?
- The critical steps: **Step 3** and **Step 4**
- **Step 3.** $\textsc{Update}(A \cap \boxplus_c, A \cap \square_c)$
  1. $\text{dist}'[u] \leftarrow \text{dist}[u]$ for all $u \in A \cap \boxplus_c$
  2. For each $v \in A \cap \square_c$, find its neighbor $u_v \in A \cap \boxplus_c$ that minimizes $\text{dist}'[u_v] + \|u_v - v\|$
  3. $\text{dist}[v] \leftarrow \min\{\text{dist}[v], \text{dist}'[u_v] + \|u_v - v\|\}$ for all $v \in A \cap \square_c$
- What if we remove the constraint that u_v is a neighbor of v?
  Then u_v is exactly the weighted nearest neighbor of v in $A \cap \boxplus_c$ (each $u \in A \cap \boxplus_c$ is assigned the weight $\text{dist}'[u]$).
  In this case, the problem can be solved by building a WVD on $A \cap \boxplus_c$.

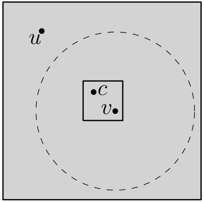
The exact SSSP algorithm

Lemma

Even if we remove the neighborhood constraint, the point u_v *we find is still a neighbor of* v.

Lemma

Even if we remove the neighborhood constraint, the point u_v we find is still a neighbor of v.



- $\text{dist}'[u] \geq \text{dist}'[c]$
- $\|u - v\| > 1 \geq \|c - v\|$
- $\implies \text{dist}'[u] + \|u - v\| > \text{dist}'[c] + \|c - v\| \implies u \neq u_v$

The exact SSSP algorithm

- **Step 3** can be done in $O(m \log m)$ time where $m = |A \cap \boxplus_c|$.

The exact SSSP algorithm

- **Step 3** can be done in $O(m \log m)$ time where $m = |A \cap \boxplus_c|$.
- How to implement **Step 4**?

The exact SSSP algorithm

- **Step 3** can be done in $O(m \log m)$ time where $m = |A \cap \boxplus_c|$.
- How to implement **Step 4**?
- Basic idea: reducing to the OIWNN problem
  Step 4 can be done in $O(m \log m + f(m))$ time where $m = |A \cap \boxplus_c|$ and $f(m)$ is the time for solving an m-operation OIWNN instance.

The exact SSSP algorithm

- **Step 3** can be done in $O(m \log m)$ time where $m = |A \cap \boxplus_c|$.
- How to implement **Step 4**?
- Basic idea: reducing to the OIWNN problem
  Step 4 can be done in $O(m \log m + f(m))$ time where $m = |A \cap \boxplus_c|$ and $f(m)$ is the time for solving an m-operation OIWNN instance.
- By showing $f(m) = O(m \log^2 m)$, we conclude the following.

Theorem

There is an SSSP algorithm on weighted UDGs using $O(n \log^2 n)$ time and $O(n)$ space, where n is the input size.

Open questions

- Improve the running time to $O(n \log n)$?
- APSP in weighted UDGs in $o(n \log^2 n)$ time?
- Can our approach be used to solve other problems in UDGs?

Thank you!

Q & A