Cluster Minimization in Geometric Graphs

Jakob Geiger
Motivation
Motivation
Cluster Minimization

Given: Geometric graph $G = (V, E)$
Cluster Minimization

Given: Geometric graph $G = (V, E)$

Goal: Find a subgraph $H = (V, E')$ of $G$ such that no two edges in $E'$ cross and the number of connected components in $H$ is minimized.
Cluster Minimization

Given: Geometric graph $G = (V, E)$

Goal: Find a subgraph $H = (V, E')$ of $G$ such that no two edges in $E'$ cross and the number of connected components in $H$ is minimized.
Edge Maximization

Given: Geometric graph $G = (V, E)$
Edge Maximization

Given: Geometric graph $G = (V, E)$

Goal: Find a subgraph $H = (V, E')$ of $G$ such that no two edges in $E'$ cross and $|E'|$ is maximized.
Edge Maximization

Given: Geometric graph $G = (V, E)$

Goal: Find a subgraph $H = (V, E')$ of $G$ such that no two edges in $E'$ cross and $|E'|$ is maximized.
# State of the art

<table>
<thead>
<tr>
<th>Problem</th>
<th>Quality</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Minimization</td>
<td>exact</td>
<td>?</td>
</tr>
<tr>
<td>– Greedy</td>
<td>?</td>
<td>polynomial</td>
</tr>
<tr>
<td>– 1-plane graphs</td>
<td>exact</td>
<td>polynomial</td>
</tr>
<tr>
<td>Edge Maximization</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

all results by [Akitaya et al. 2019]
### My contribution

<table>
<thead>
<tr>
<th>Problem</th>
<th>Quality</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Min.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
<tr>
<td>– Greedy</td>
<td>no const. factor</td>
<td>$n + k + m \log m$</td>
</tr>
<tr>
<td>– Rev. Greedy</td>
<td>no const. factor</td>
<td>$n + k \log k + m \log m$</td>
</tr>
<tr>
<td>– 1-plane graphs</td>
<td>exact</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Edge Max.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization

- Given an instance of Independent Set,
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization

- Given an instance of Independent Set,
- Construct an equivalent L-shape intersection graph...

[Gonçalves et al. 2018]
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization
- Given an instance of Independent Set,
- Construct an equivalent L-shape intersection graph...
- ... then construct an equivalent segment intersection graph.

[Biedl 2020]
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization

- Given an instance of Independent Set,
- Construct an equivalent L-shape intersection graph...
- ... then construct an equivalent segment intersection graph.
- Use the segments as edges in a geometric graph and place vertices at each endpoint.
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization
- Given an instance of Independent Set,
- Construct an equivalent L-shape intersection graph...
- ... then construct an equivalent segment intersection graph.
- Use the segments as edges in a geometric graph and place vertices at each endpoint.
- In the resulting geometric graph, a solution with $2n - k$ clusters represents an independent set of size $k$. 
Heuristics

Greedy: Iteratively select the least crossed edge
Heuristics

Greedy: Iteratively select the least crossed edge

Reverse Greedy: Iteratively delete the most crossed edge
Heuristics

Greedy: Iteratively select the least crossed edge

Reverse Greedy: Iteratively delete the most crossed edge

Preprocessing: compute all edge crossings
Heuristics

Greedy: Iteratively select the least crossed edge

Reverse Greedy: Iteratively delete the most crossed edge

Preprocessing: compute all edge crossings

⇒ $O(k + m \log m)$ [Balaban 1995]
Greedy

Iteratively select the least crossed edge
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers $\Rightarrow$ Fibonacci-Heap!
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters ⇒ $O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers ⇒ Fibonacci-Heap!

<table>
<thead>
<tr>
<th></th>
<th>(O(\log n)^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td></td>
</tr>
<tr>
<td>ExtractMin</td>
<td>(O(\log n)^*)</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>(O(1)^*)</td>
</tr>
</tbody>
</table>

*amortized
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters \( \Rightarrow O(n + m\alpha(m)) \)

Use Priority Queue to manage current crossing numbers

\( \Rightarrow \) Fibonacci-Heap! \( \Rightarrow O(k + m \log m) \)
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Fibonacci-Heap! $\Rightarrow O(k + m \log m)$

Overall Runtime: $O(n + k + m \log m)$
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Fibonacci-Heap! $\Rightarrow O(k + m \log m)$

Overall Runtime: $O(n + k + m \log m)$

1-plane graphs: $m, k \in O(n)$
Greedy

Iteratively select the least crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Fibonacci-Heap! $\Rightarrow O(k + m \log m)$

Overall Runtime: $O(n + k + m \log m)$

1-plane graphs: $m, k \in O(n)$

$\Rightarrow$ Overall runtime reduces to $O(n \log n)$!
Reverse Greedy

Iteratively delete the most crossed edge
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters \( \Rightarrow O(n + m\alpha(m)) \)
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters ⇒ $O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers ⇒ Fibonacci-Heap!
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Fibonacci-Heap!
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers $\Rightarrow$ Binary Search Tree
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Binary Search Tree

<table>
<thead>
<tr>
<th></th>
<th>$O(\log n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>REMOVE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>EXTRACTMIN</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DECREASEKEY</strong></td>
<td></td>
</tr>
</tbody>
</table>
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers $\Rightarrow$ Binary Search Tree $\Rightarrow O(k \log k + m \log m)$
Reverse Greedy

Iteratively delete the most crossed edge

Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers
$\Rightarrow$ Binary Search Tree $\Rightarrow O(k \log k + m \log m)$

Overall Runtime: $O(n + k \log k + m \log m)$
Performance Analysis – Theoretical

$n$ red, $n + 1$ blue vertices
Performance Analysis – Theoretical

$n$ red, $n + 1$ blue vertices
Performance Analysis – Theoretical

$n$ red, $n + 1$ blue vertices
Greedy/Reverse Greedy: $n + 2$ clusters

$n$ red, $n + 1$ blue vertices
Performance Analysis – Theoretical

Greedy/Reverse Greedy: $n + 2$ clusters

Optimal solution:

$n$ red, $n + 1$ blue vertices
Performance Analysis – Theoretical

Greedy/Reverse Greedy: \( n + 2 \) clusters

Optimal solution: 4 clusters

\[ \Rightarrow \text{no constant approximation factor for both heuristics!} \]
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy

A_i, B_i, C_i, D_i, A'_i, B'_i, C'_i
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy

\[ A_i, A'_i, B_i, B'_i, C_i, C'_i, D_i \]
Performance – Greedy vs. Reverse Greedy
Performance – Greedy vs. Reverse Greedy

Greedy 7 clusters vs. Reverse Greedy $k+7$!
An ILP for Cluster Minimization

Sketch:
An ILP for Cluster Minimization

Sketch:

- Model Cluster Minimization as a flow network.
An ILP for Cluster Minimization

Sketch:

- Model Cluster Minimization as a flow network.
- Each node is either a source or a sink.
An ILP for Cluster Minimization

Sketch:

• Model Cluster Minimization as a flow network.
• Each node is either a source or a sink.
• Each edge is either selected or not selected, crossed edges are mutually exclusive.
An ILP for Cluster Minimization

Sketch:

- Model Cluster Minimization as a flow network.
- Each node is either a source or a sink.
- Each edge is either selected or not selected, crossed edges are mutually exclusive.
- Selected edges may transport flow, unselected edges may not.
An ILP for Cluster Minimization

Sketch:

• Model Cluster Minimization as a flow network.
• Each node is either a source or a sink.
• Each edge is either selected or not selected, crossed edges are mutually exclusive.
• Selected edges may transport flow, unselected edges may not.
• Each sink represents the "center" of a cluster, connected nodes send the generated flow there.
An ILP for Cluster Minimization

Sketch:

• Model Cluster Minimization as a flow network.
• Each node is either a source or a sink.
• Each edge is either selected or not selected, crossed edges are mutually exclusive.
• Selected edges may transport flow, unselected edges may not.
• Each sink represents the "center" of a cluster, connected nodes send the generated flow there.
• ILP minimizes the number of sinks.
Experiment setup

- Use map of places of interest in a city.
Experiment setup

- Use map of places of interest in a city.
- Divide the map in quadrants of varying sizes.
Experiment setup

• Use map of places of interest in a city.
• Divide the map in quadrants of varying sizes.
• Connect the vertices with $\beta$-skeletons.
Experiment setup

- Use map of places of interest in a city.
- Divide the map in quadrants of varying sizes.
- Connect the vertices with $\beta$-skeletons.
Experiment setup

- Use map of places of interest in a city.
- Divide the map in quadrants of varying sizes.
- Connect the vertices with $\beta$-skeletons.
- Run both heuristics, ILP where feasible.

\[ \beta = 0.5 \]
\[ \beta = 0.9 \]
Experiment setup

$\beta = 0.5$

50 points, 15 clusters
Performance Analysis - Experiments

\[ \beta = 0.5 \]
Performance Analysis - Experiments

\[ \beta = 0.9 \]
Experiment Summary

Biggest difference: Greedy 37 clusters vs. ILP 34 clusters!
Experiment Summary

Biggest difference: Greedy 37 clusters vs. ILP 34 clusters!

Reverse Greedy tends to perform better than Greedy, but differences are marginal
## Summary and Future Work

<table>
<thead>
<tr>
<th>Problem</th>
<th>Quality</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Min.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
<tr>
<td>– Greedy</td>
<td>no const. factor</td>
<td>$n + k + m \log m$</td>
</tr>
<tr>
<td>– Rev. Greedy</td>
<td>no const. factor</td>
<td>$n + k \log k + m \log m$</td>
</tr>
<tr>
<td>– 1-plane graphs</td>
<td>exact</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Edge Max.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>
Summary and Future Work

<table>
<thead>
<tr>
<th>Problem</th>
<th>Quality</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Min.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
<tr>
<td>– Greedy</td>
<td>no const. factor</td>
<td>$n + k + m \log m$</td>
</tr>
<tr>
<td>– Rev. Greedy</td>
<td>no const. factor</td>
<td>$n + k \log k + m \log m$</td>
</tr>
<tr>
<td>– 1-plane graphs</td>
<td>exact</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Edge Max.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

- There is a graph family on which the Greedy algorithm is arbitrarily better than the Reverse Greedy algorithm.
- Is there a graph family where the opposite is true?
Summary and Future Work

<table>
<thead>
<tr>
<th>Problem</th>
<th>Quality</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Min.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
<tr>
<td>– Greedy</td>
<td>no const. factor</td>
<td>$n + k + m \log m$</td>
</tr>
<tr>
<td>– Rev. Greedy</td>
<td>no const. factor</td>
<td>$n + k \log k + m \log m$</td>
</tr>
<tr>
<td>– 1-plane graphs</td>
<td>exact</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Edge Max.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

• There is a graph family on which the Greedy algorithm is arbitrarily better than the Reverse Greedy algorithm.

• Is there a graph family where the opposite is true?

• Is there a constant factor approximation for Cluster Minimization?
## Summary and Future Work

<table>
<thead>
<tr>
<th>Problem</th>
<th>Quality</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Min.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
<tr>
<td>– Greedy</td>
<td>no const. factor</td>
<td>$n + k + m \log m$</td>
</tr>
<tr>
<td>– Rev. Greedy</td>
<td>no const. factor</td>
<td>$n + k \log k + m \log m$</td>
</tr>
<tr>
<td>– 1-plane graphs</td>
<td>exact</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>Edge Max.</td>
<td>exact</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

- There is a graph family on which the Greedy algorithm is arbitrarily better than the Reverse Greedy algorithm.
- Is there a graph family where the opposite is true?
- Is there a constant factor approximation for Cluster Minimization?
- How does the problem change if we allow some crossings?
Summary and Future Work

• Can we enhance the Greedy algorithm somehow?