Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

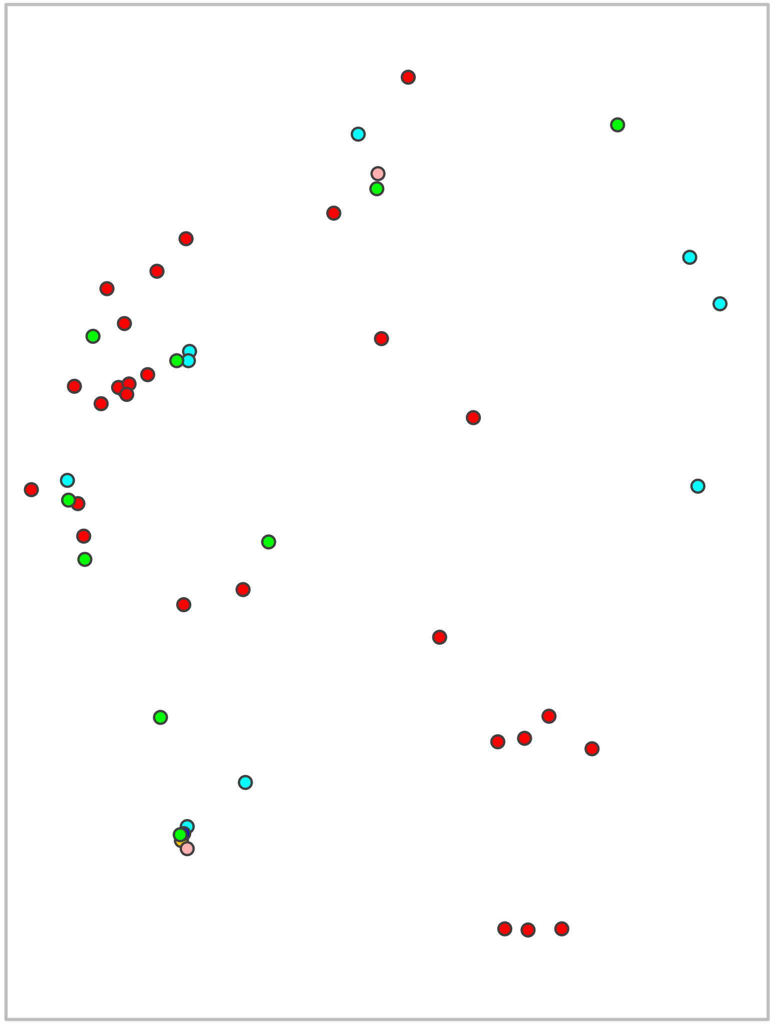
Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

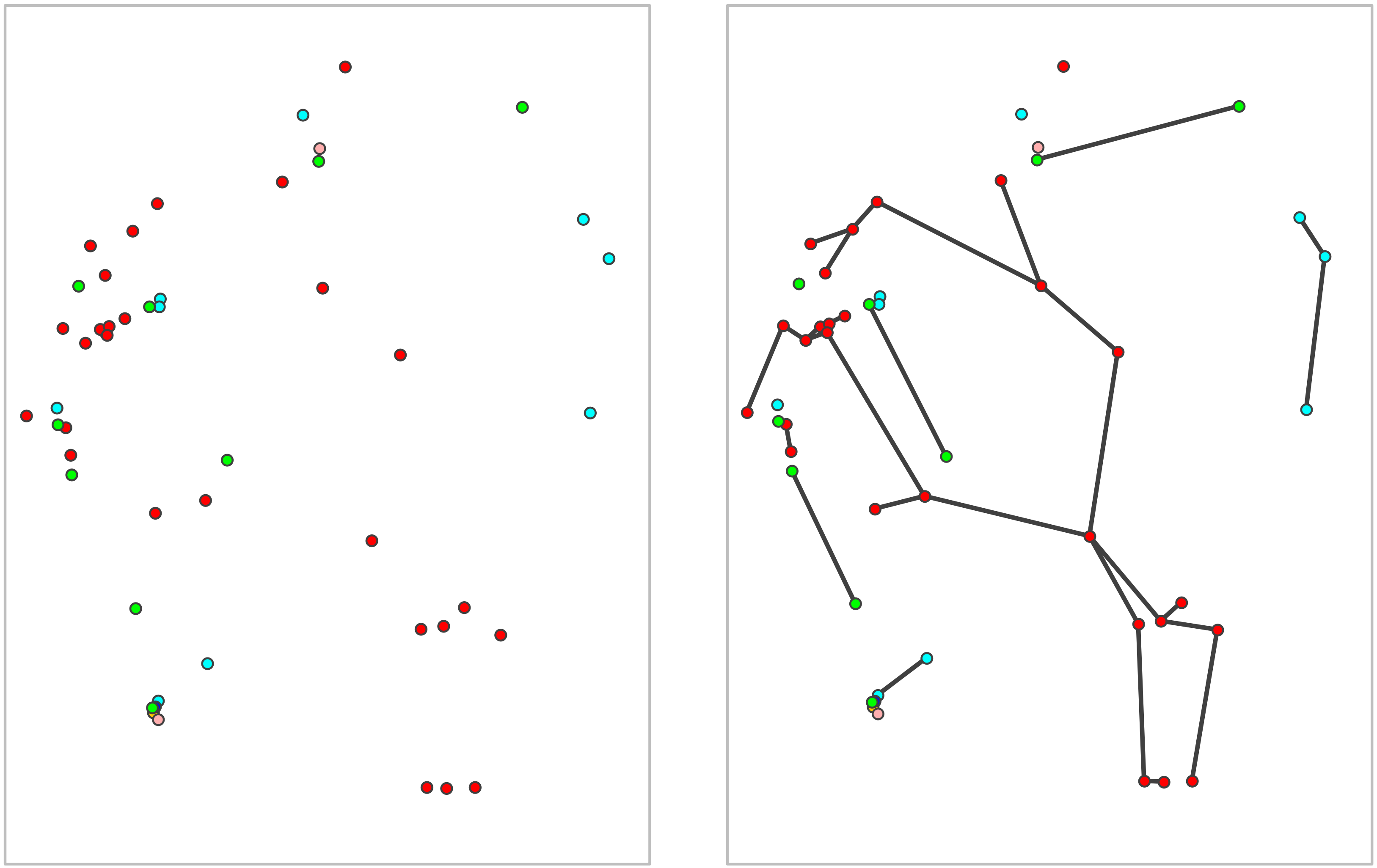
Cluster Minimization in Geometric Graphs

Jakob Geiger

Motivation

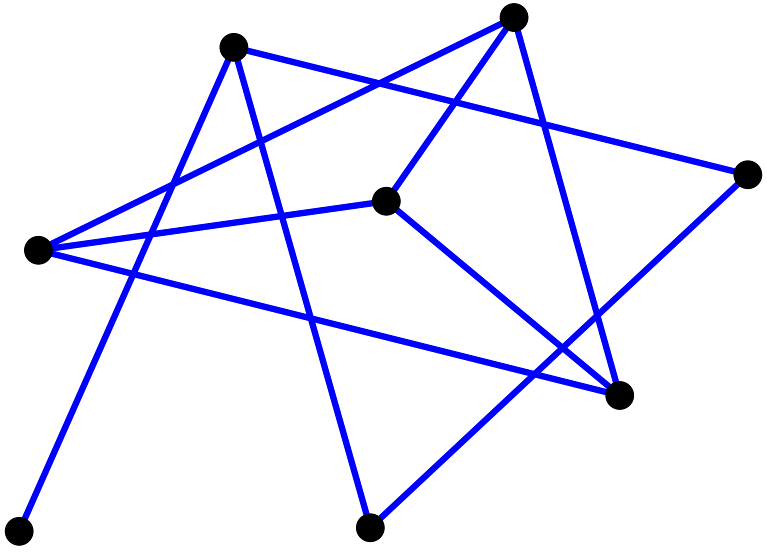


Motivation



Cluster Minimization

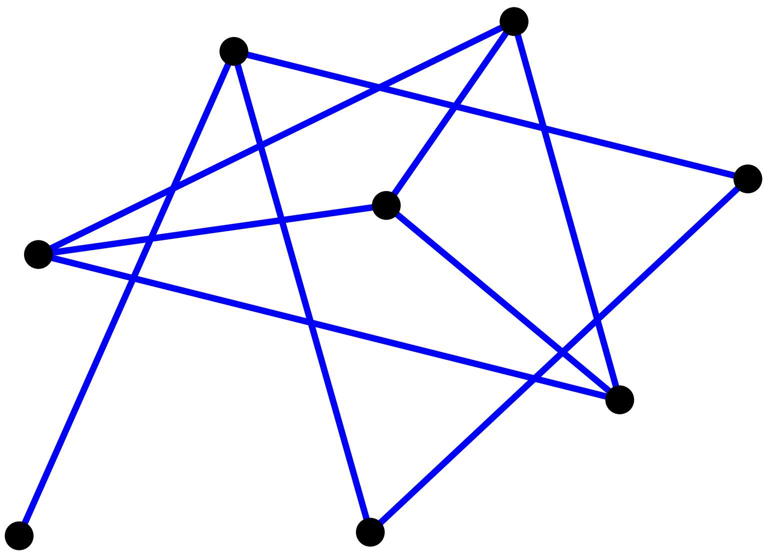
Given: Geometric graph $G = (V, E)$



Cluster Minimization

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

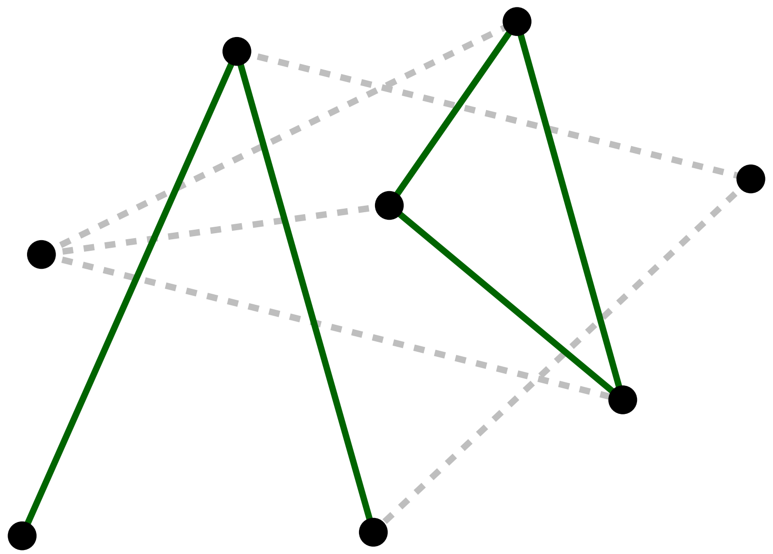
Goal: Find a subgraph $H = (V, E')$ of G such that no two edges in E' cross and the number of connected components in H is minimized.



Cluster Minimization

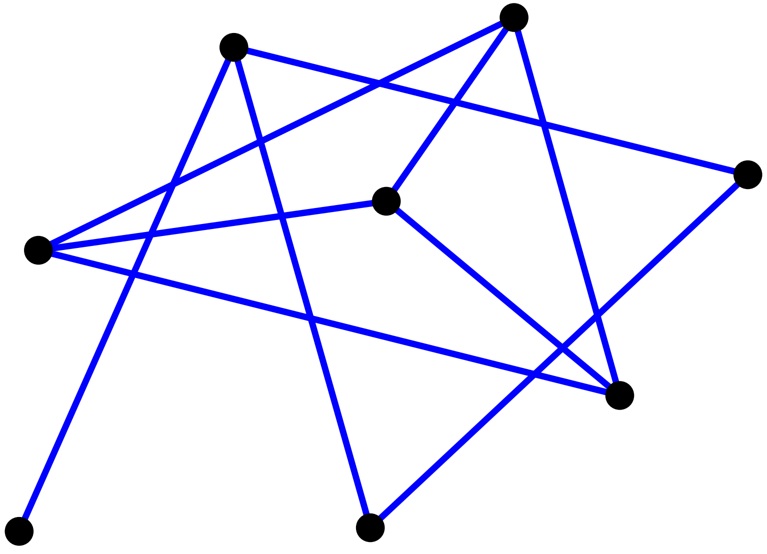
Given: Geometric graph $G = (V, E)$

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

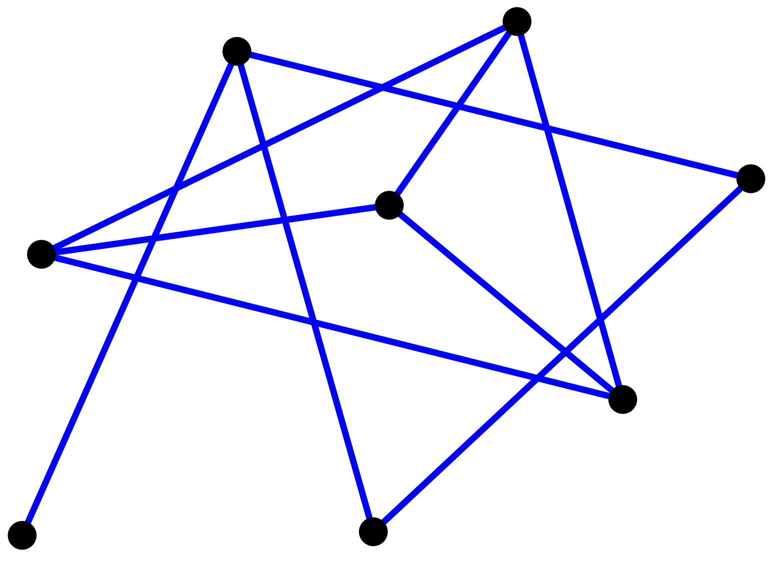
Given: Geometric graph $G = (V, E)$



Edge Maximization

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

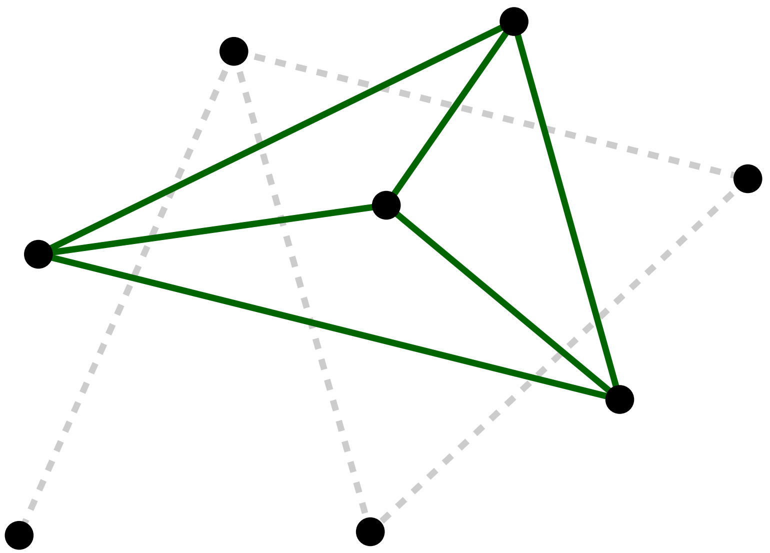
Goal: Find a subgraph $H = (V, E')$ of G such that no two edges in E' cross and $|E'|$ is maximized.



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

Problem	Quality	Runtime
Cluster Minimization	exact	?
– Greedy	?	polynomial
– 1-plane graphs	exact	polynomial
Edge Maximization	exact	NP-hard

all results by [Akitaya et al. 2019]

My contribution

Problem	Quality	Complexity
Cluster Min.	exact	NP-hard
– Greedy	no const. factor	$n + k + m \log m$
– Rev. Greedy	no const. factor	$n + k \log k + m \log m$
– 1-plane graphs	exact	$n \log n$
Edge Max.	exact	NP-hard

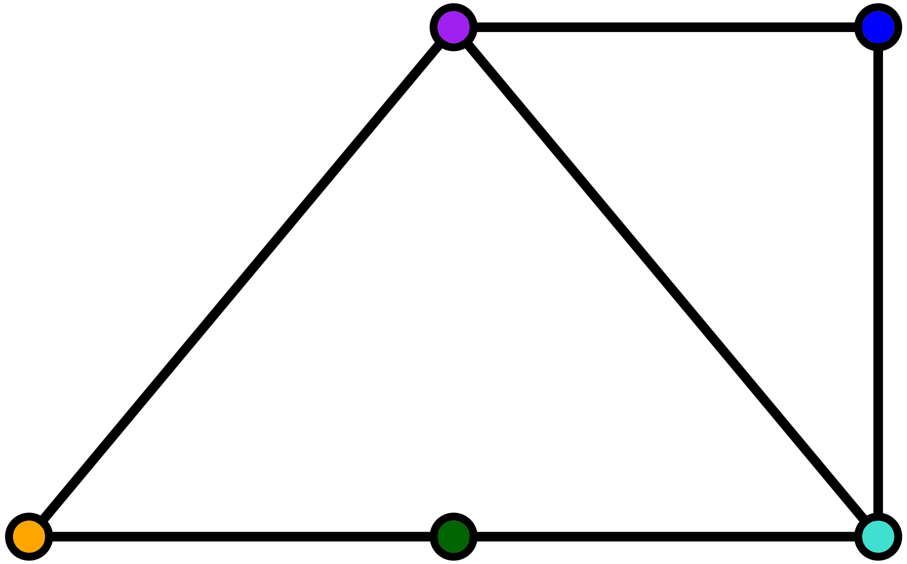
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization

NP-Hardness

Independent Set $\leq_p$ Cluster Minimization

- Given an instance of Independent Set,

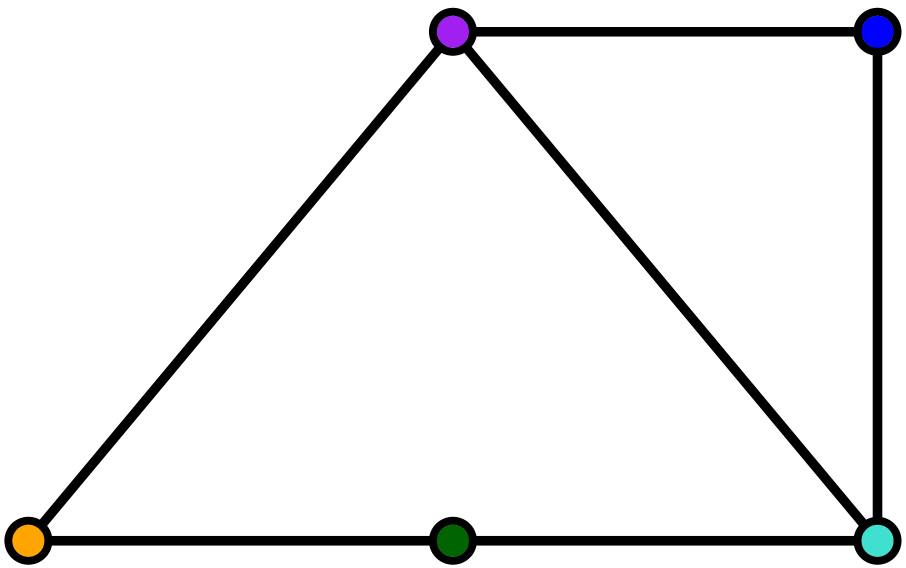


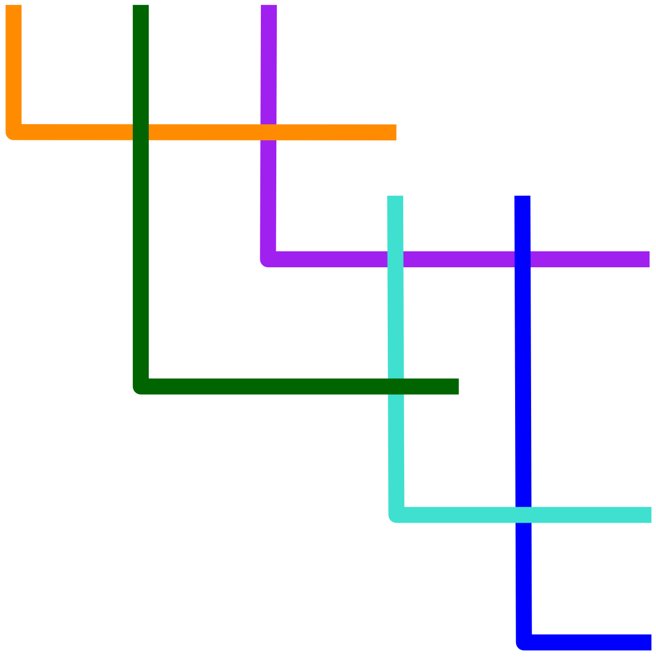
NP-Hardness

Independent Set $\leq_p$ Cluster Minimization

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

[Gonçalves et al. 2018]



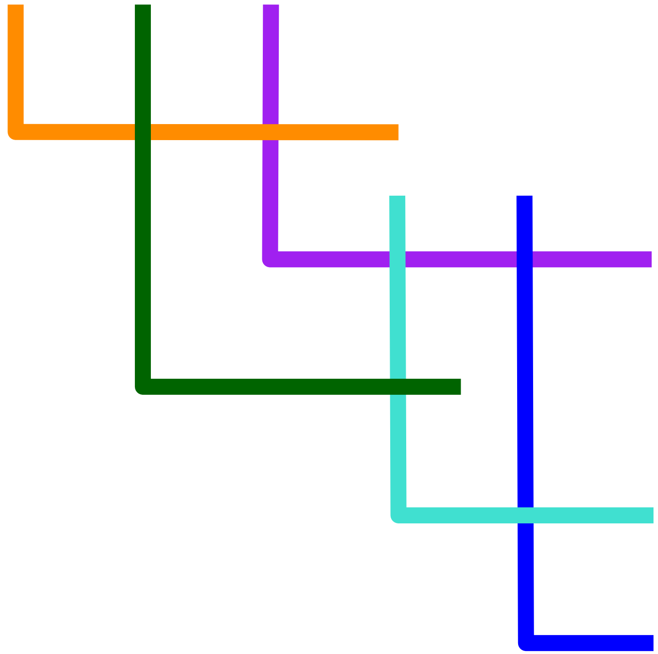


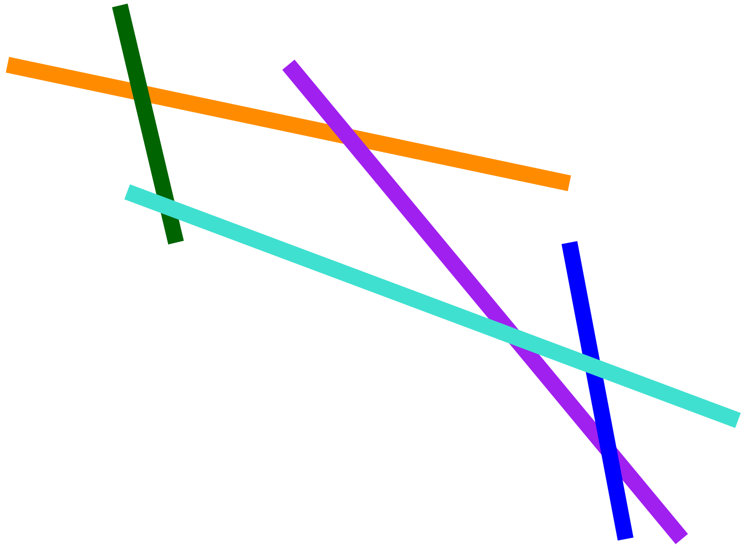
NP-Hardness

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]

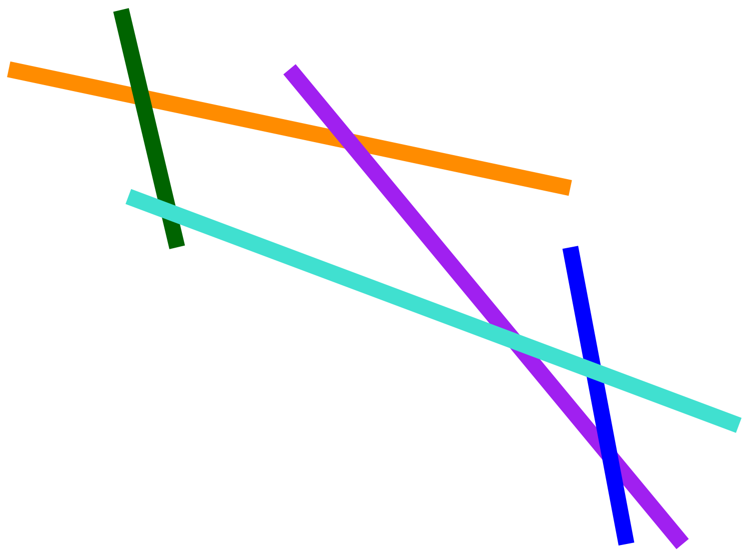


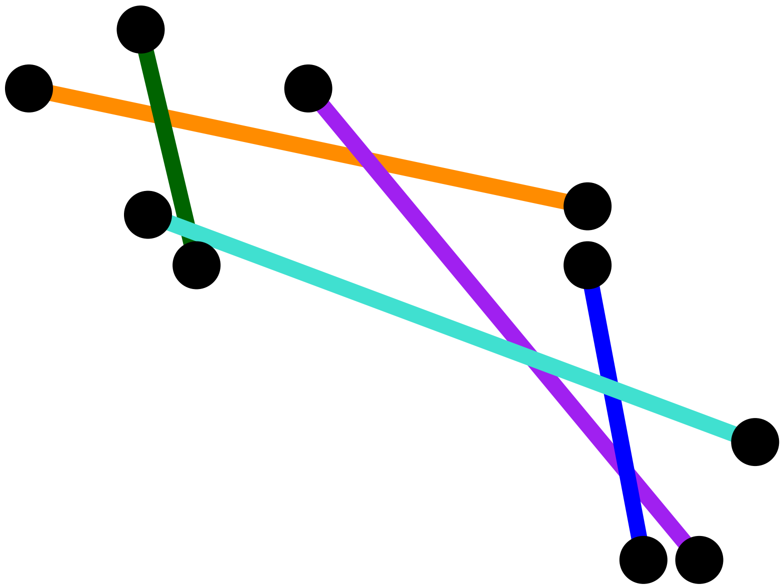


NP-Hardness

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.

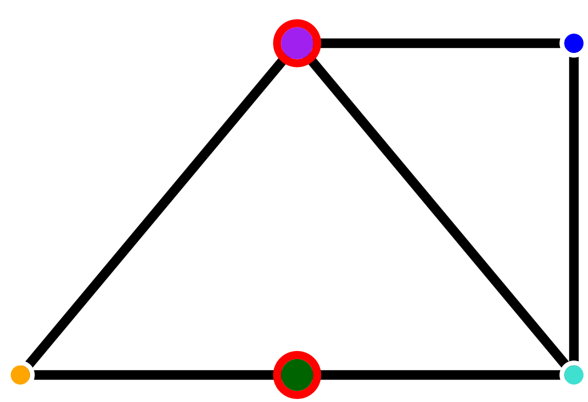


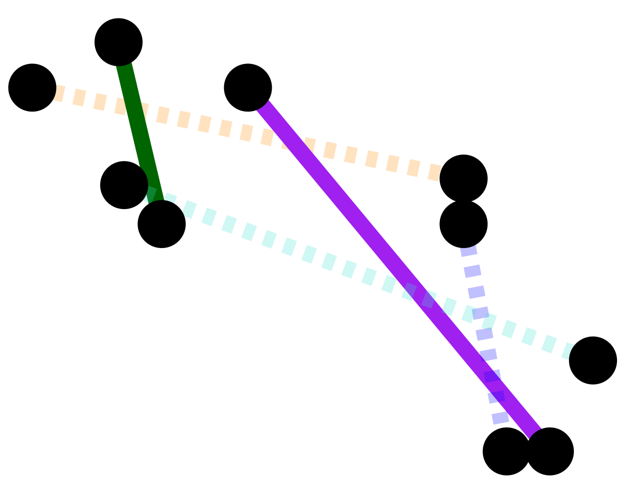


NP-Hardness

Independent Set $\leq_{\mathrm{p}}$ 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

$\Rightarrow O(k + m \log m)$ [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 $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Fibonacci-Heap!

Remove	$O(\log n)^*$
ExtractMin	$O(\log n)^*$
DecreaseKey	$O(1)^*$

*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! $\quad \Rightarrow O(k + m \log m)$

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 $\Rightarrow O(n + m\alpha(m))$

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

$\Rightarrow$ ~~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

Remove	$O(\log n)$
ExtractMin	$O(\log n)$
DecreaseKey	$O(\log n)$

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

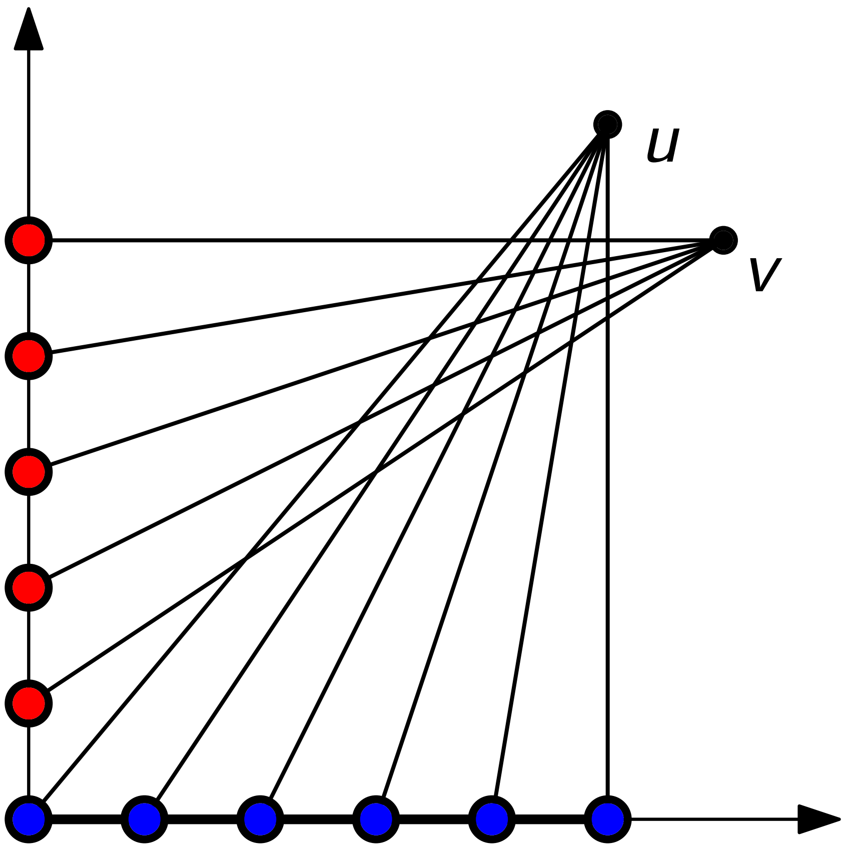
Use Union-Find to manage clusters $\Rightarrow O(n + m\alpha(m))$

Use Priority Queue to manage current crossing numbers

$\Rightarrow$ Binary Search Tree $\Rightarrow O(k \log k + m \log m)$

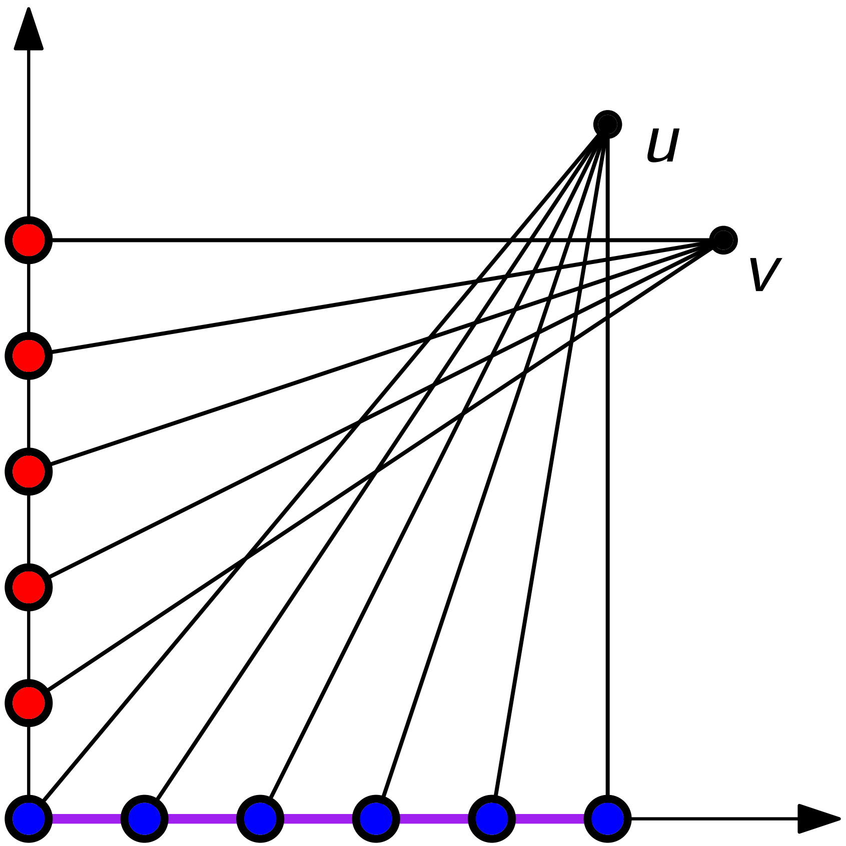
Overall Runtime: $O(n + k \log k + m \log m)$

Performance Analysis – Theoretical



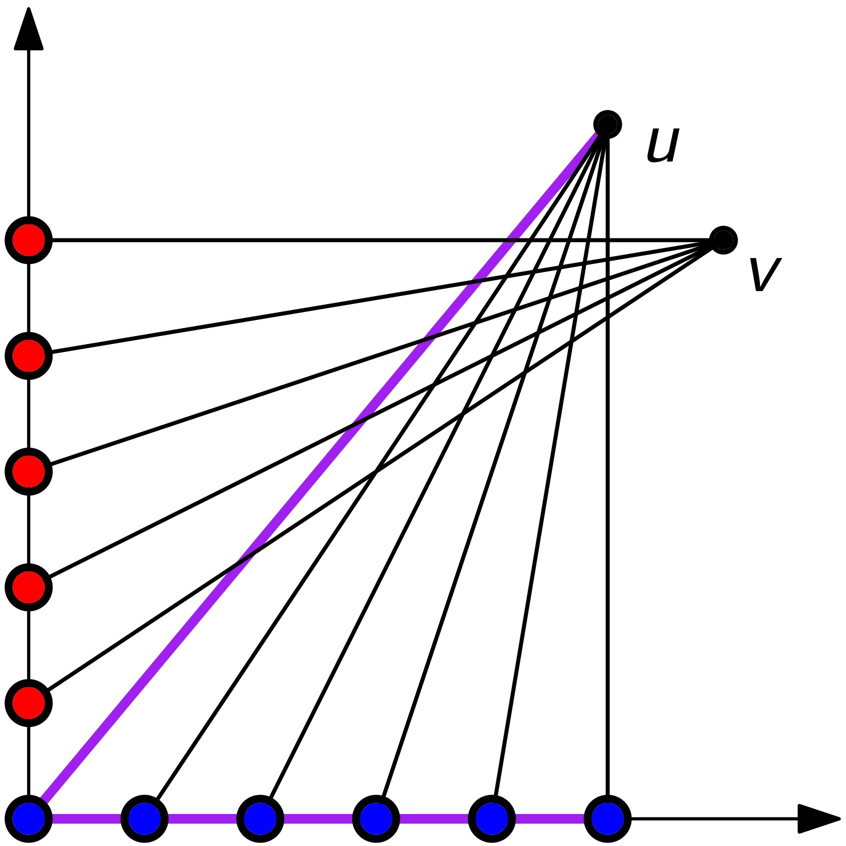
n red, $n+1$ blue vertices

Performance Analysis – Theoretical



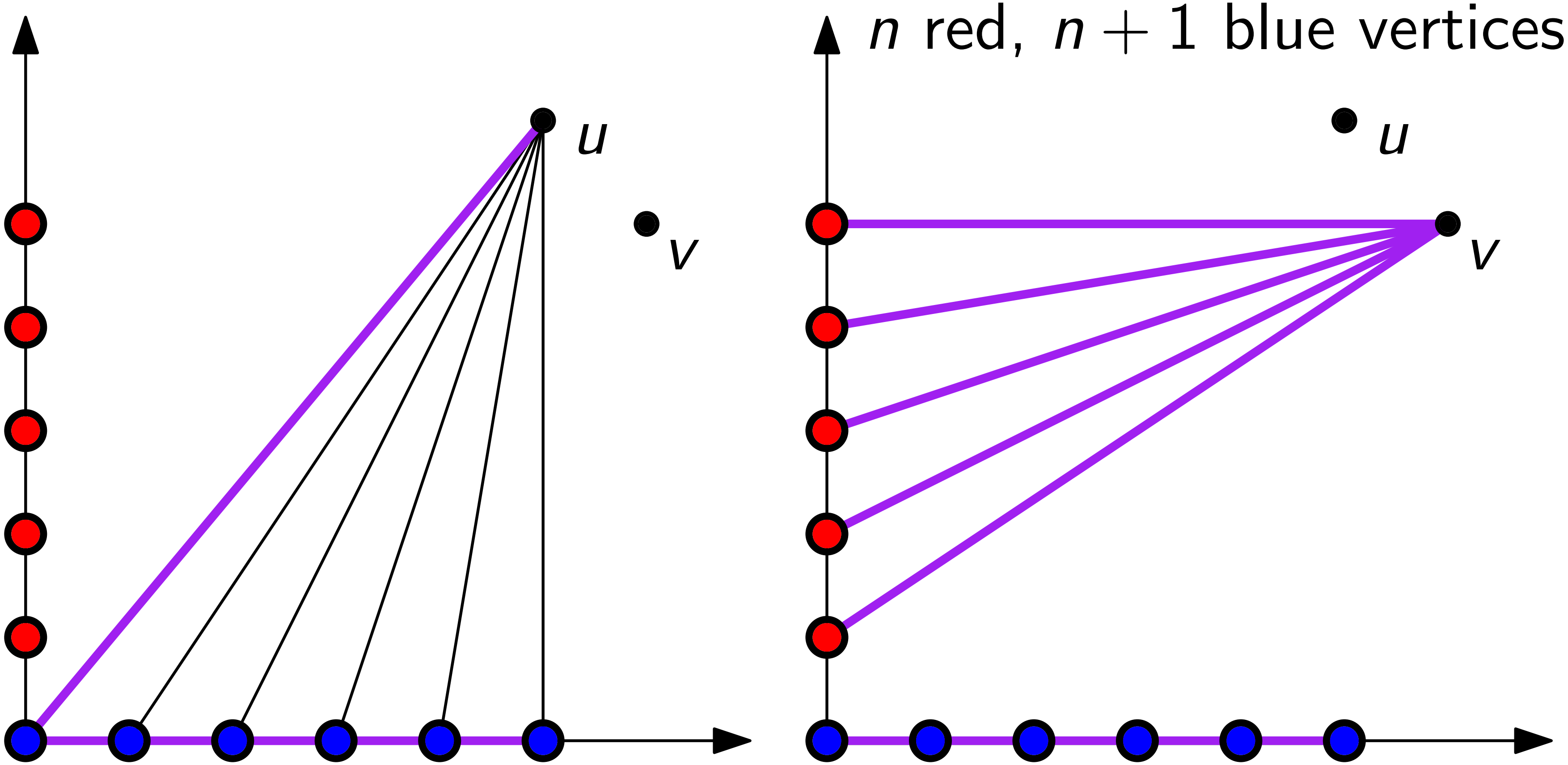
n red, $n+1$ blue vertices

Performance Analysis – Theoretical



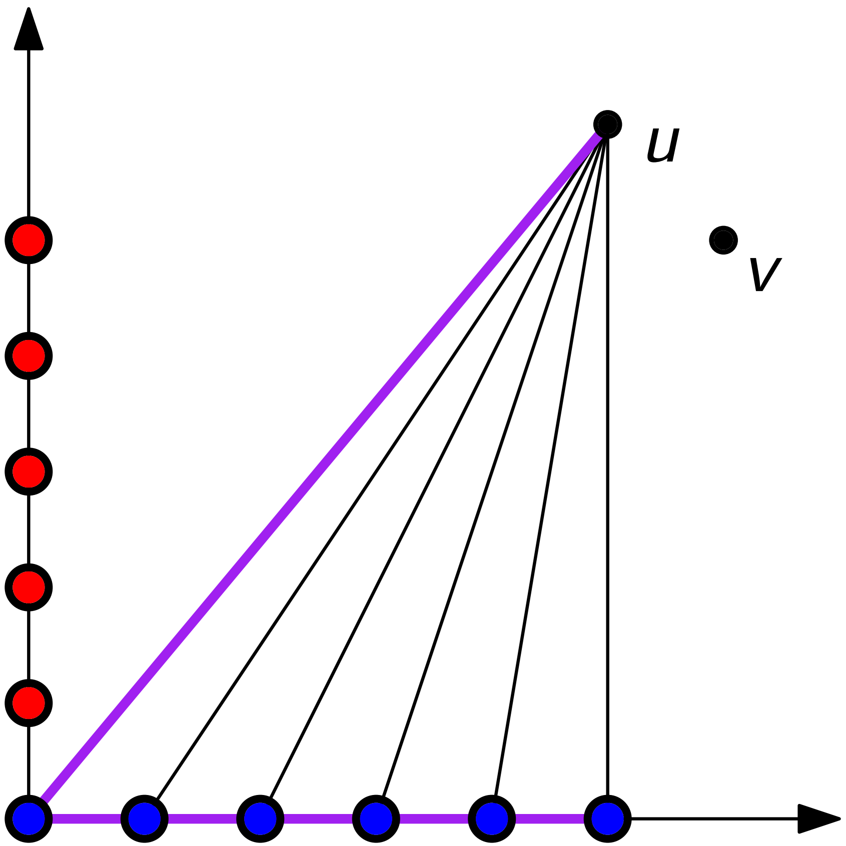
n red, $n+1$ blue vertices

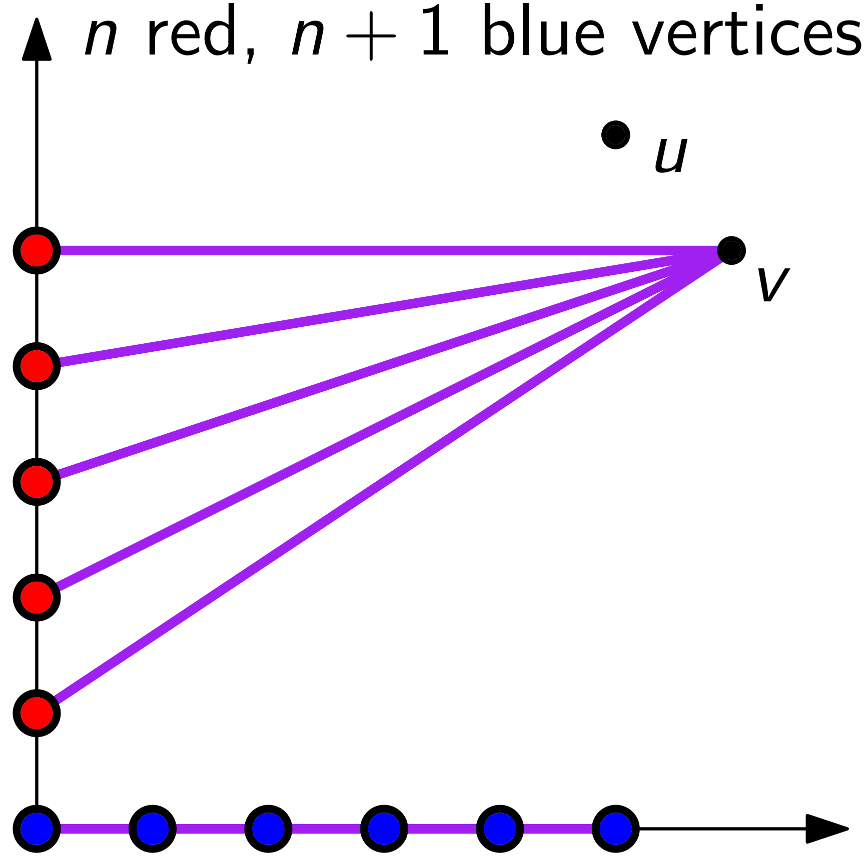
Performance Analysis – Theoretical



Greedy/Reverse Greedy: $n+2$ clusters

Performance Analysis – Theoretical

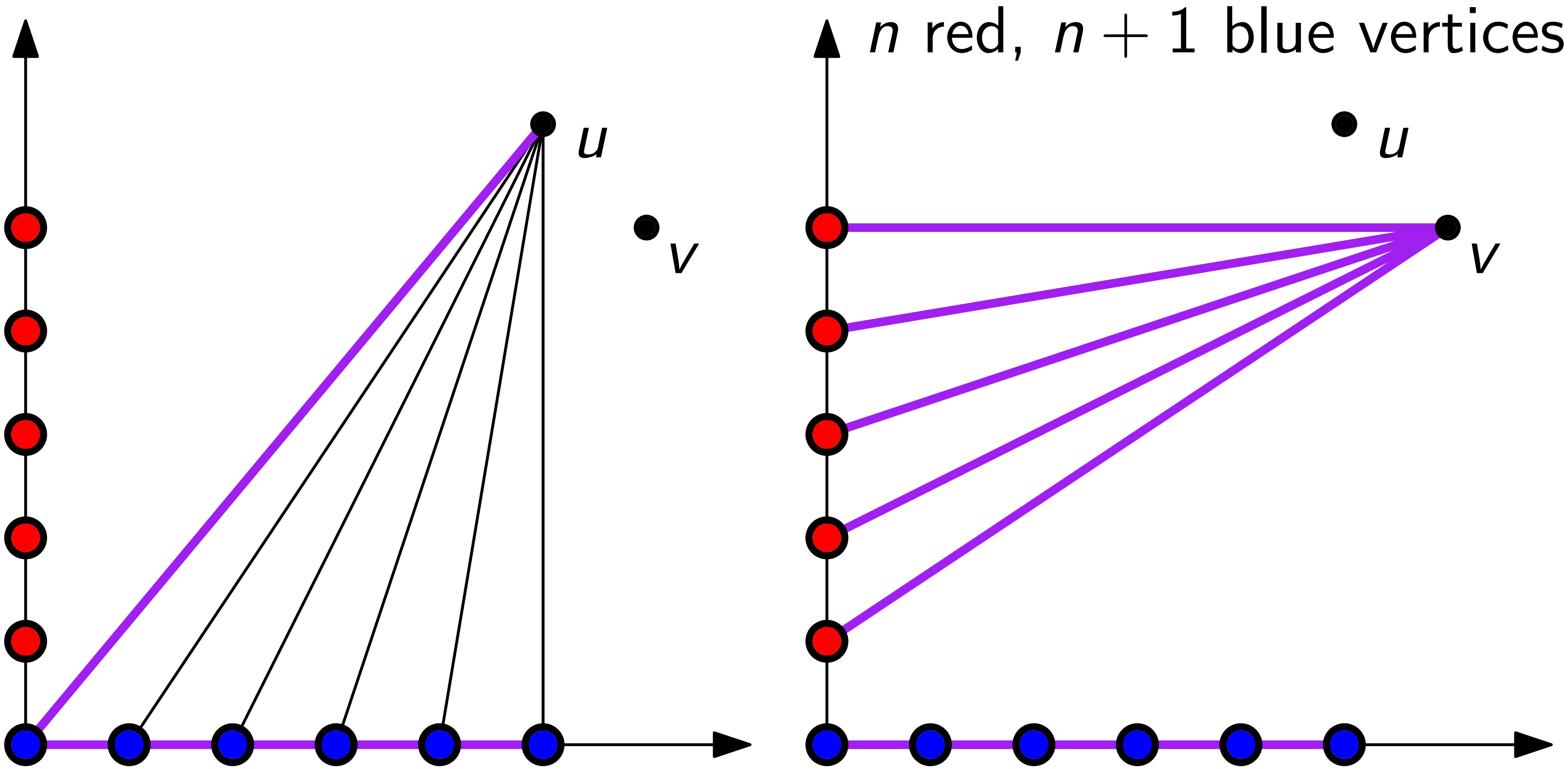




Greedy/Reverse Greedy: $n+2$ clusters

Optimal solution:

Performance Analysis – Theoretical

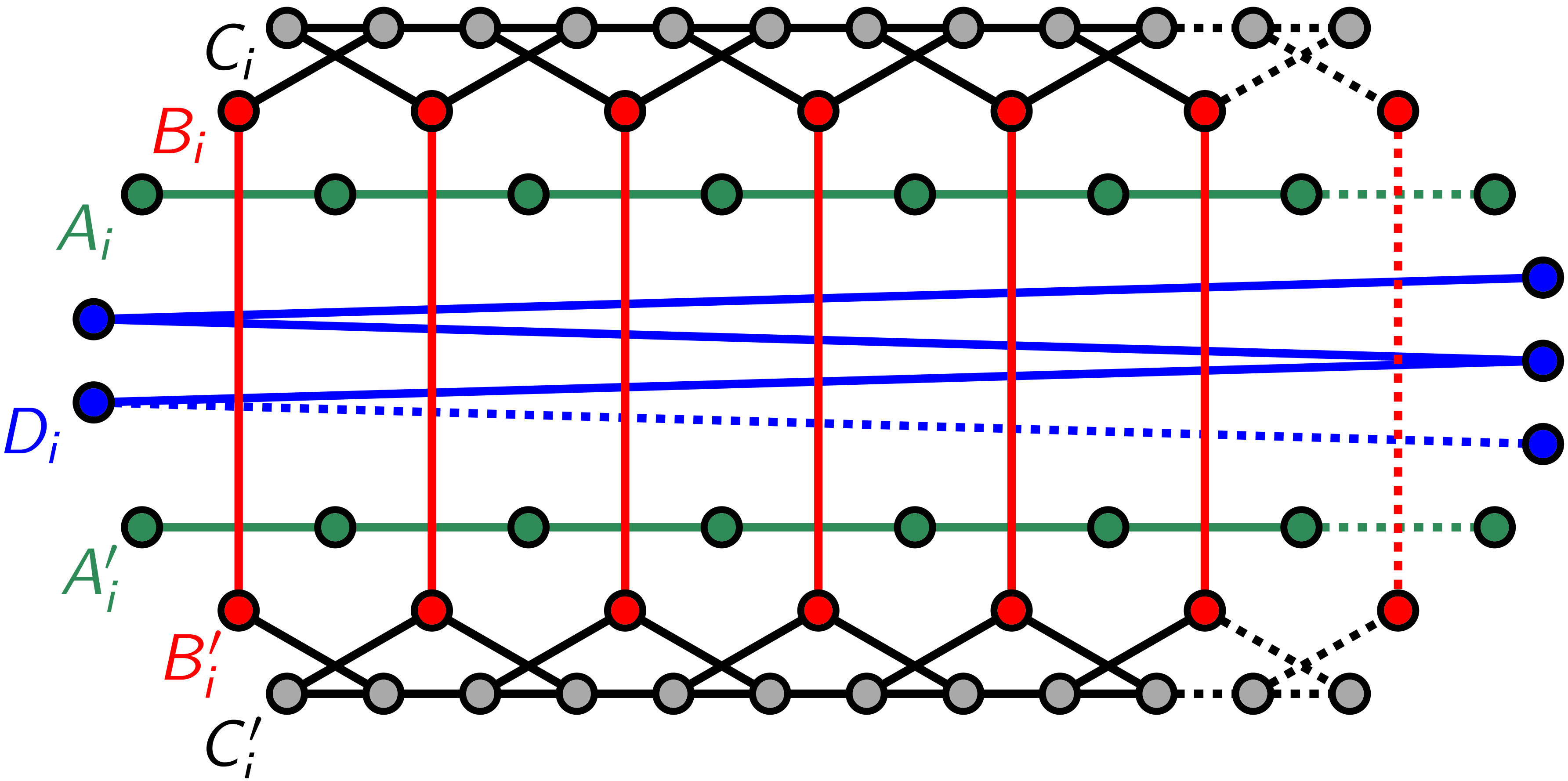


Greedy/Reverse Greedy: $n+2$ clusters

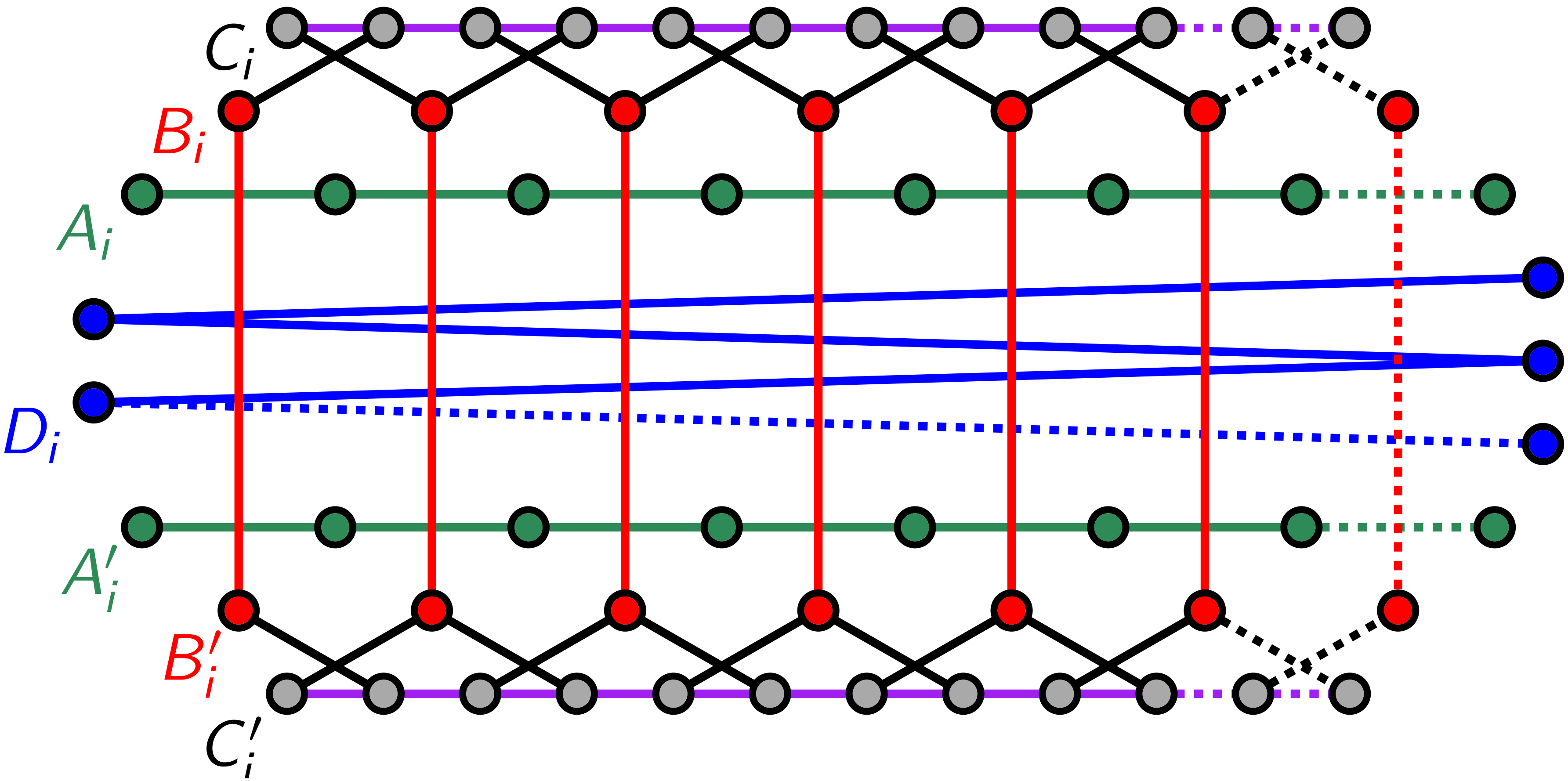
Optimal solution: 4 clusters

$\Rightarrow$ no constant approximation factor for both heuristics!

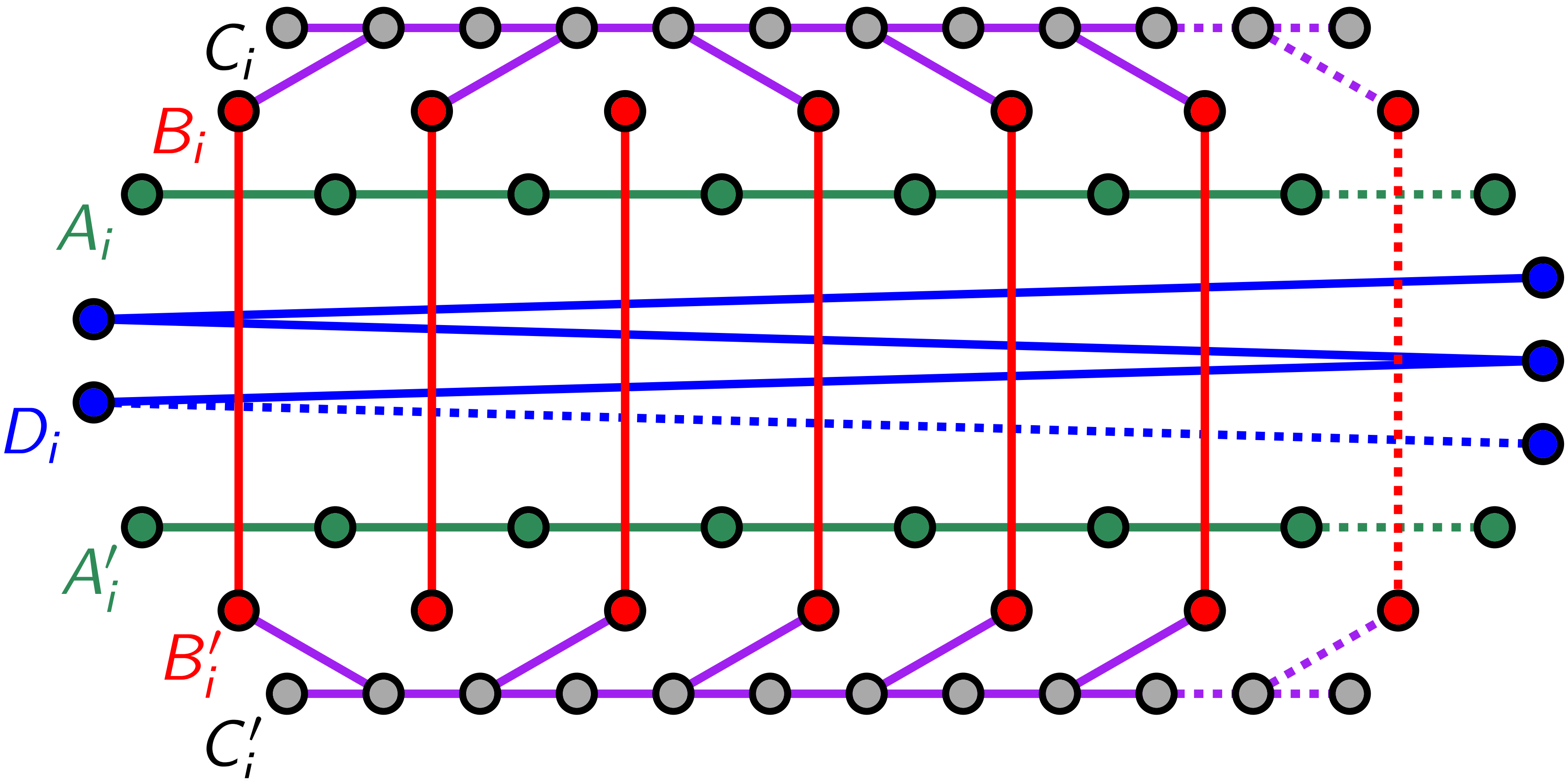
Performance – Greedy vs. Reverse Greedy



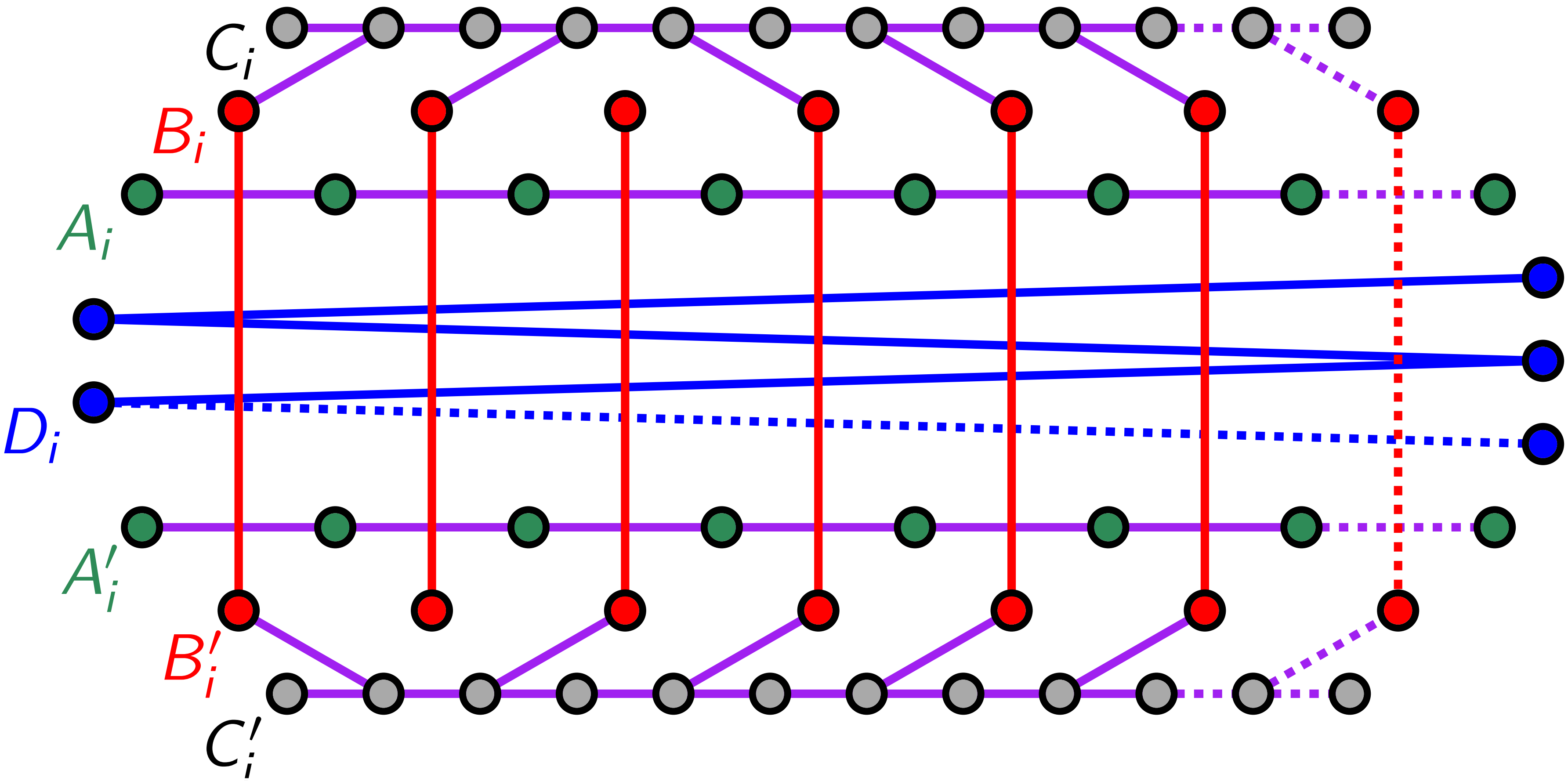
Performance – Greedy vs. Reverse Greedy



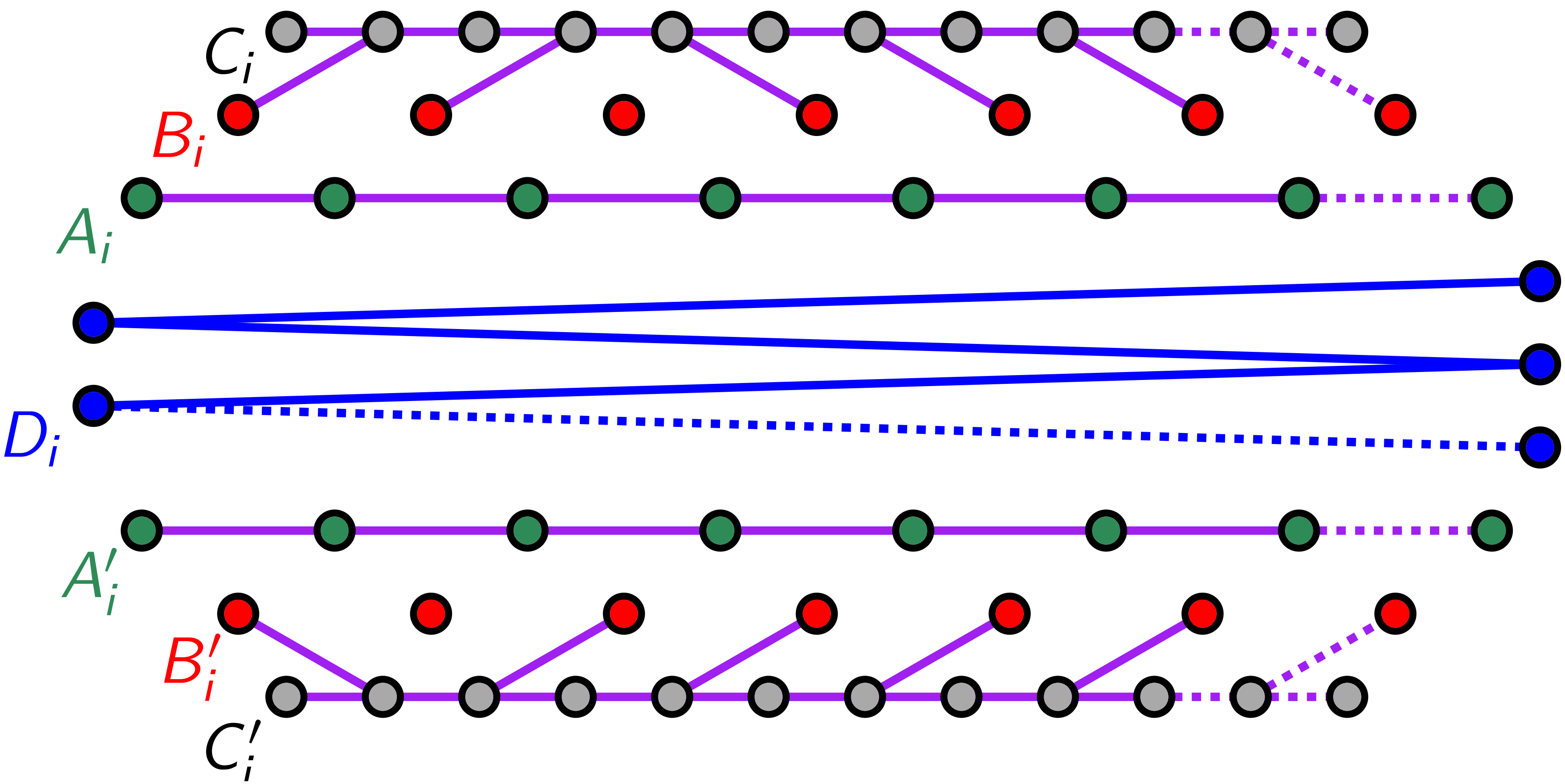
Performance – Greedy vs. Reverse Greedy



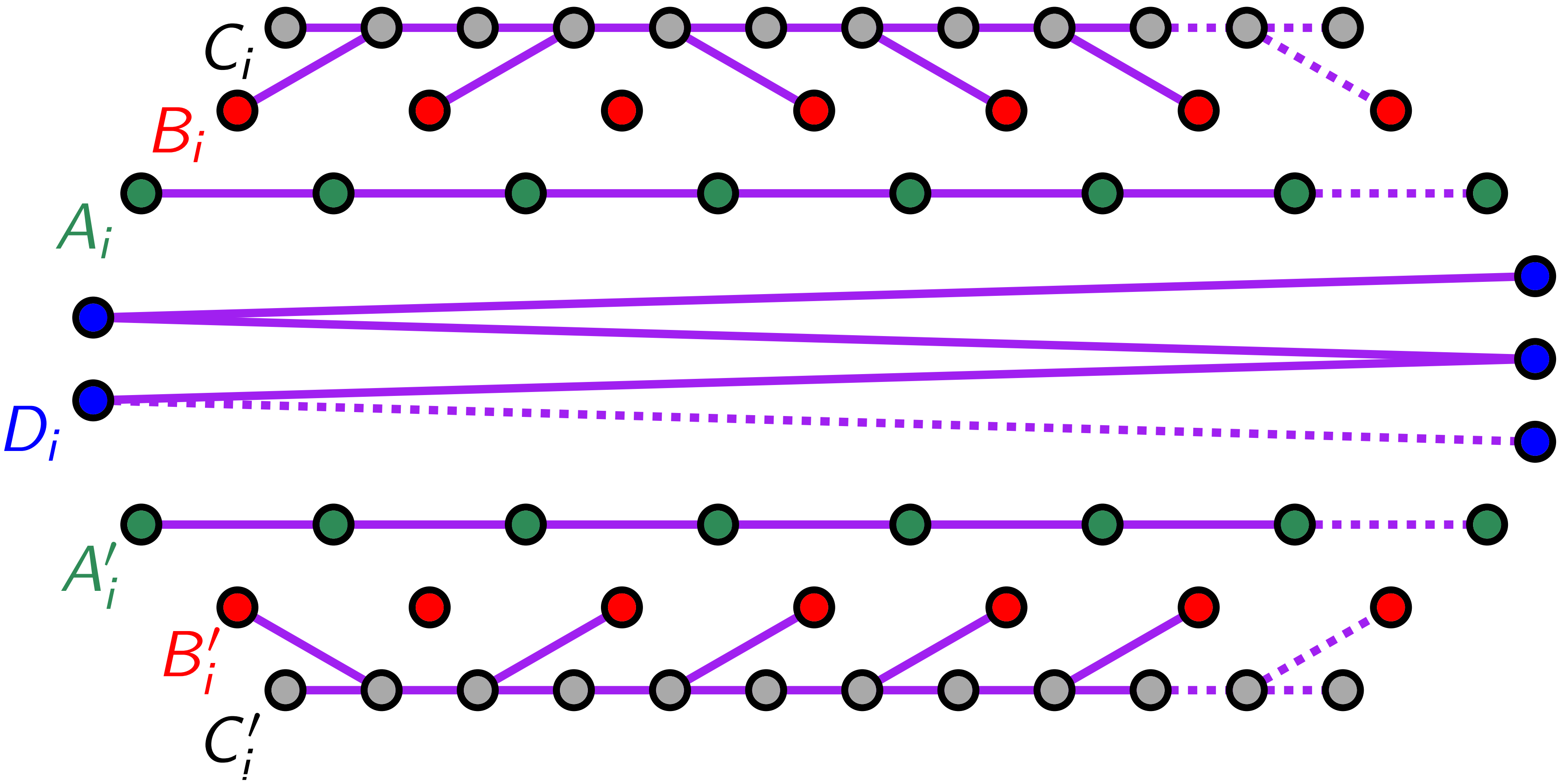
Performance – Greedy vs. Reverse Greedy



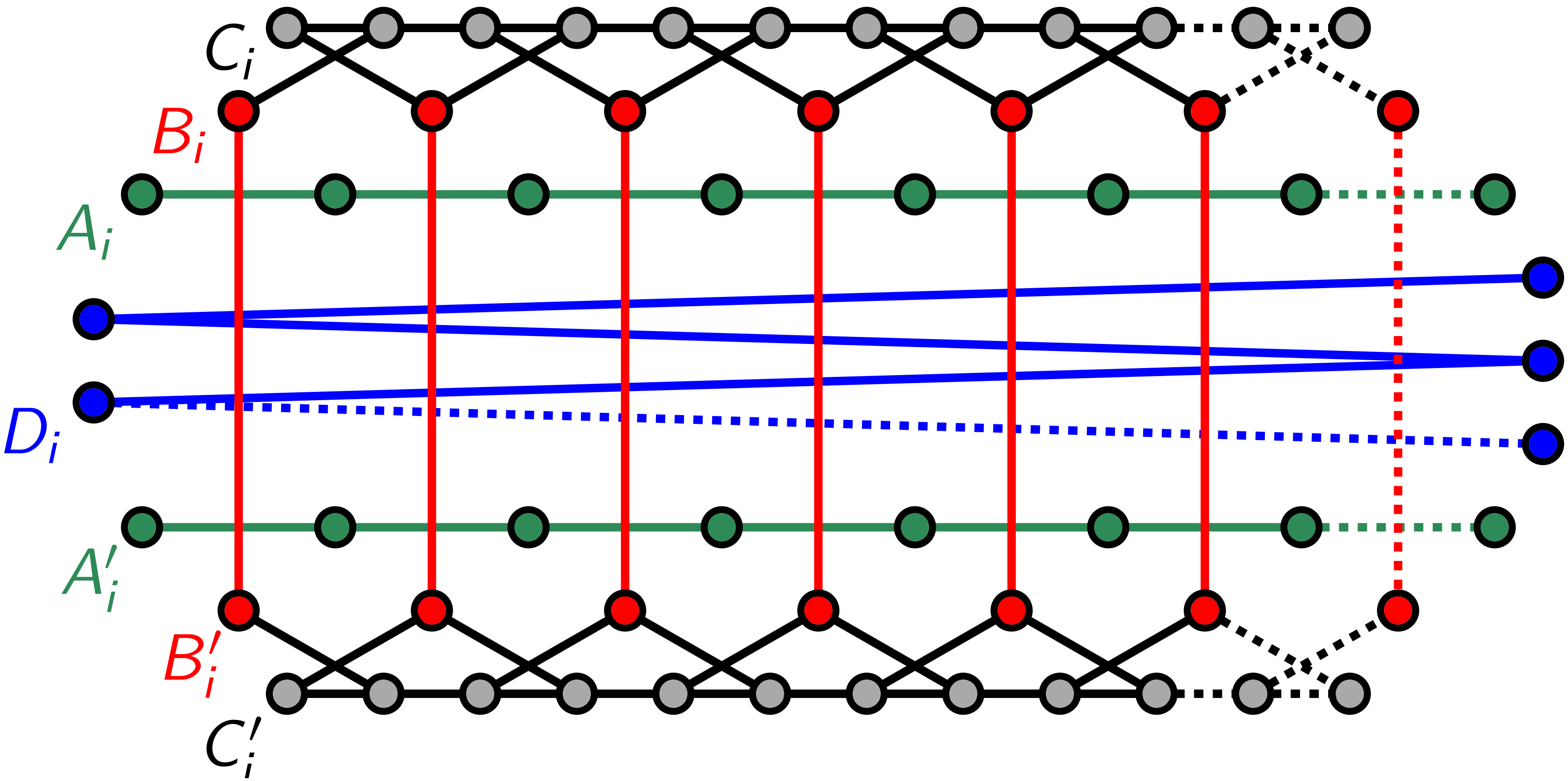
Performance – Greedy vs. Reverse Greedy



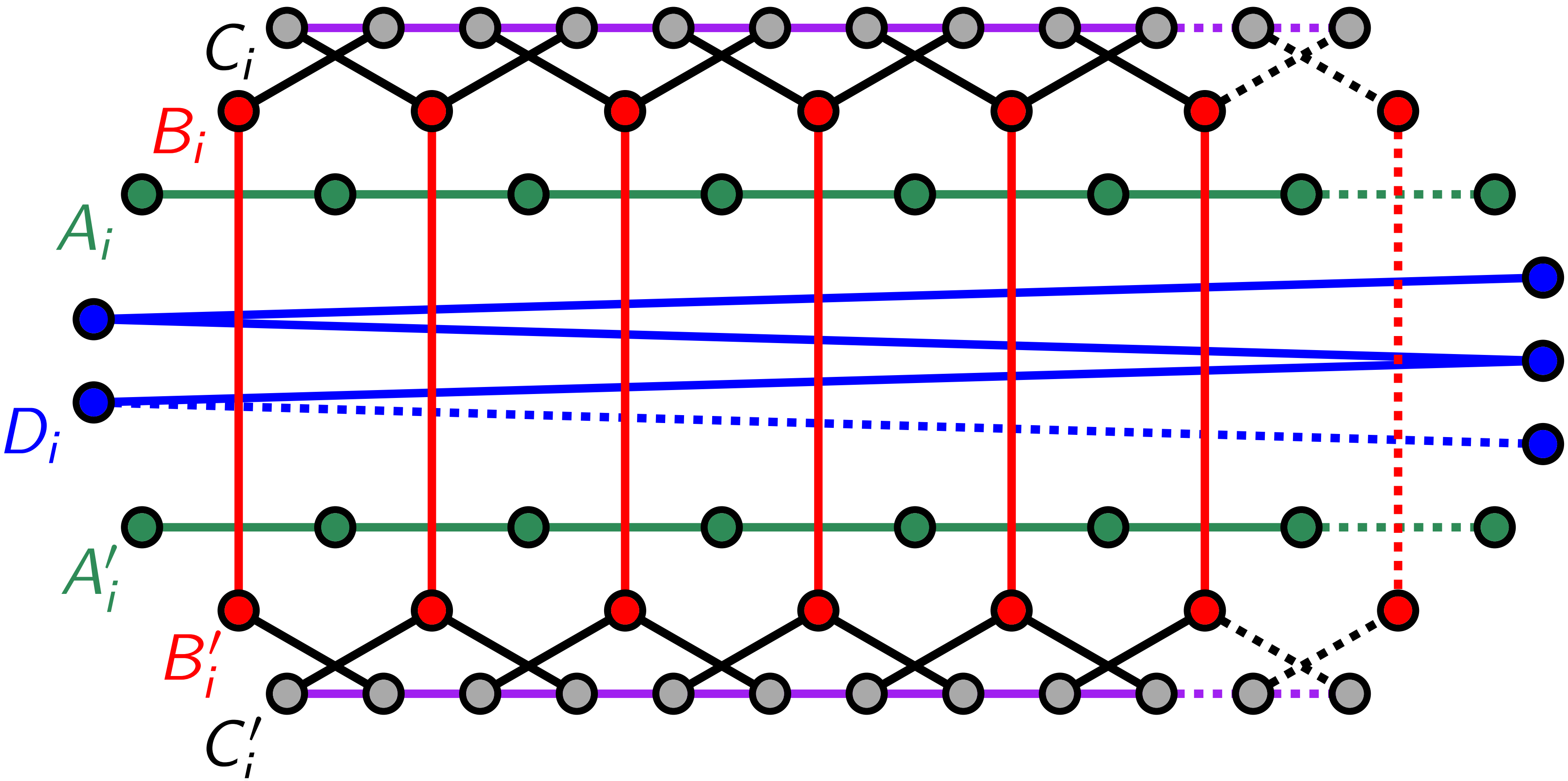
Performance – Greedy vs. Reverse Greedy



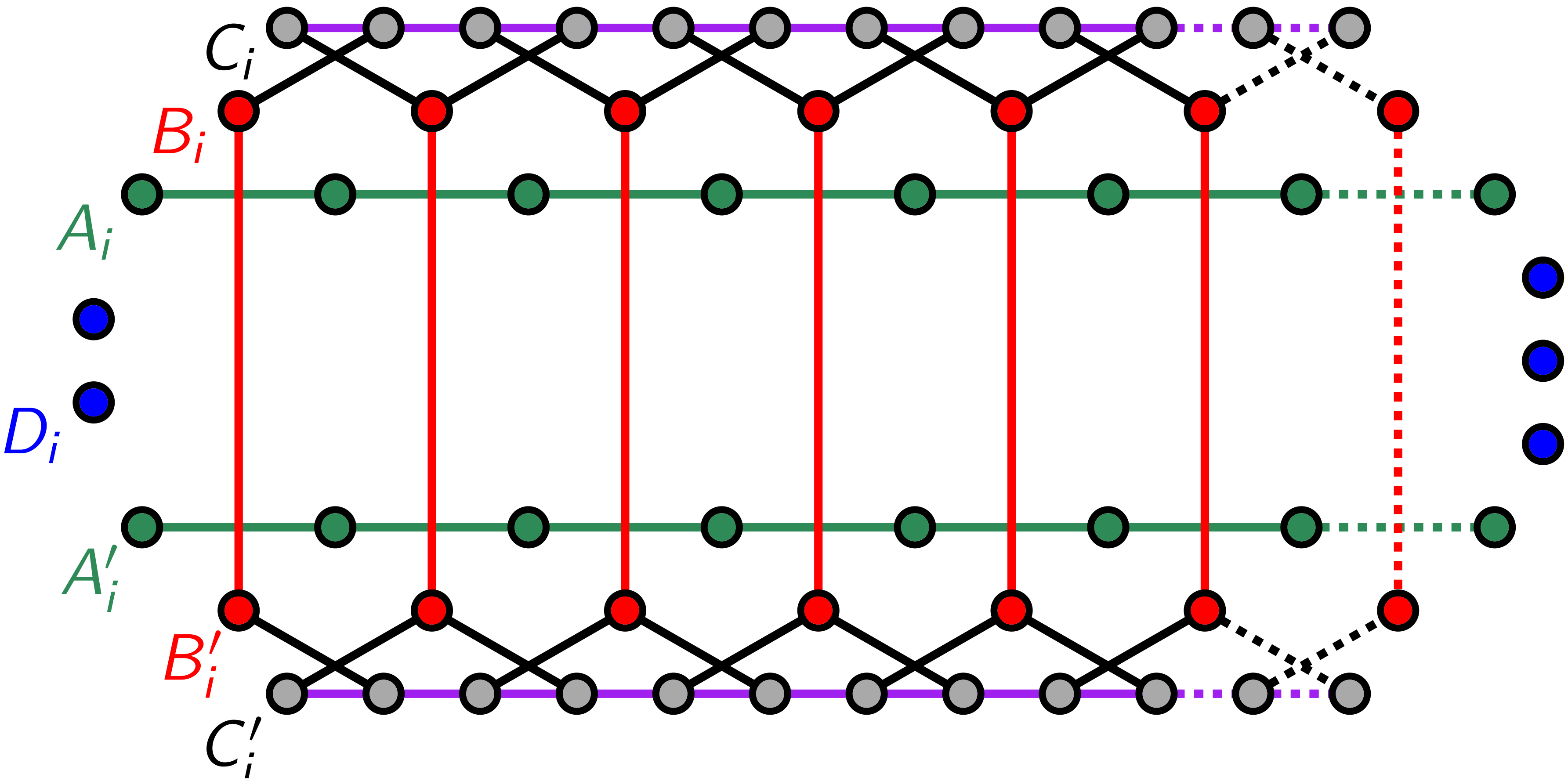
Performance – Greedy vs. Reverse Greedy



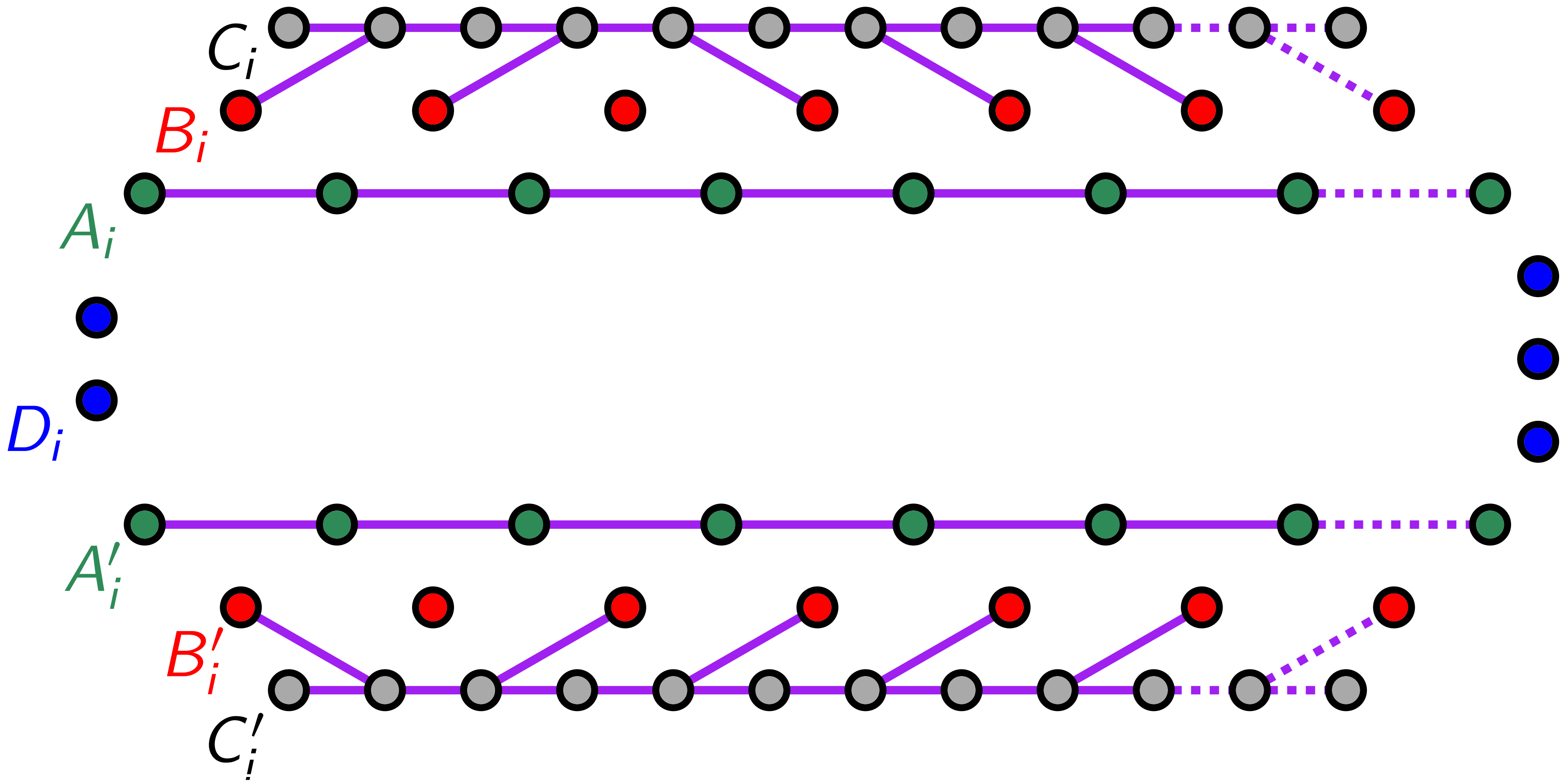
Performance – Greedy vs. Reverse Greedy



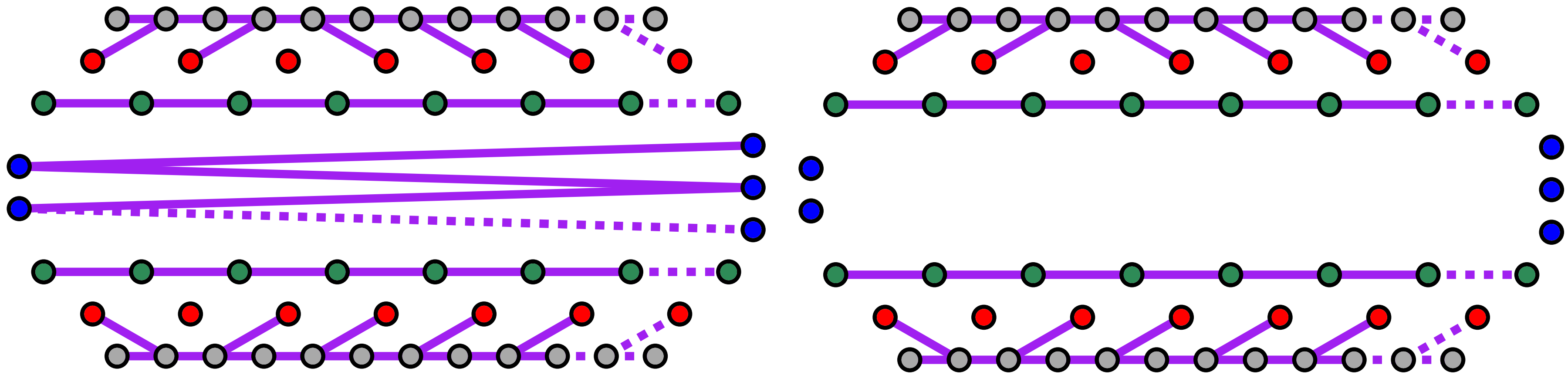
Performance – Greedy vs. Reverse Greedy



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

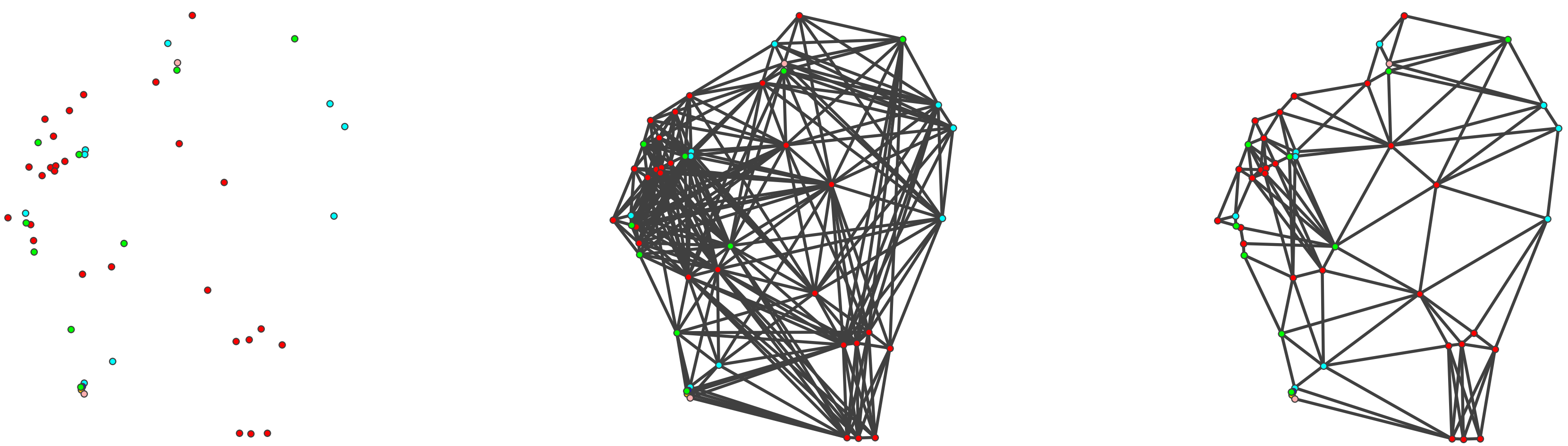
- Use map of places of interest in a city.
- Divide the map in quadrants of varying sizes.

Experiment setup

- Use map of places of interest in a city.
- Divide the map in quadrants of varying sizes.
- Connect the vertices with β-skeletons.

Experiment setup

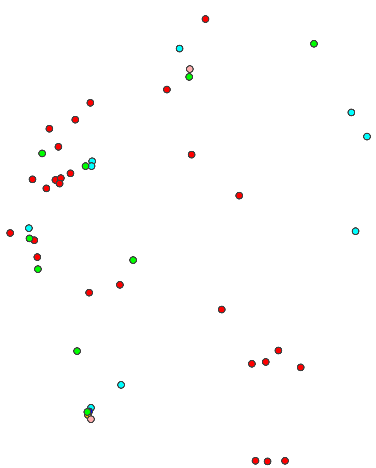
- Use map of places of interest in a city.
- Divide the map in quadrants of varying sizes.
- Connect the vertices with β-skeletons.

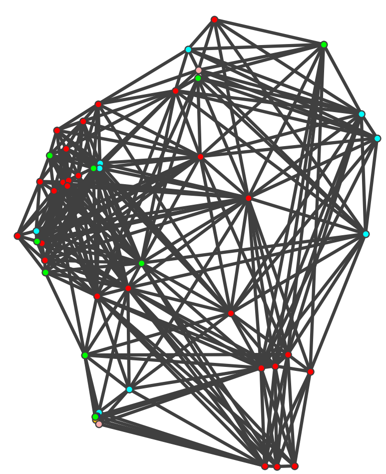


$\beta = 0.5$ $\beta = 0.9$

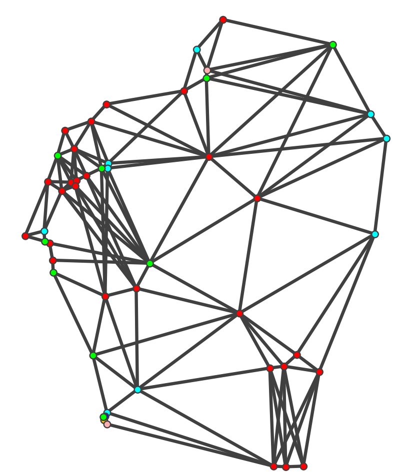
Experiment setup

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



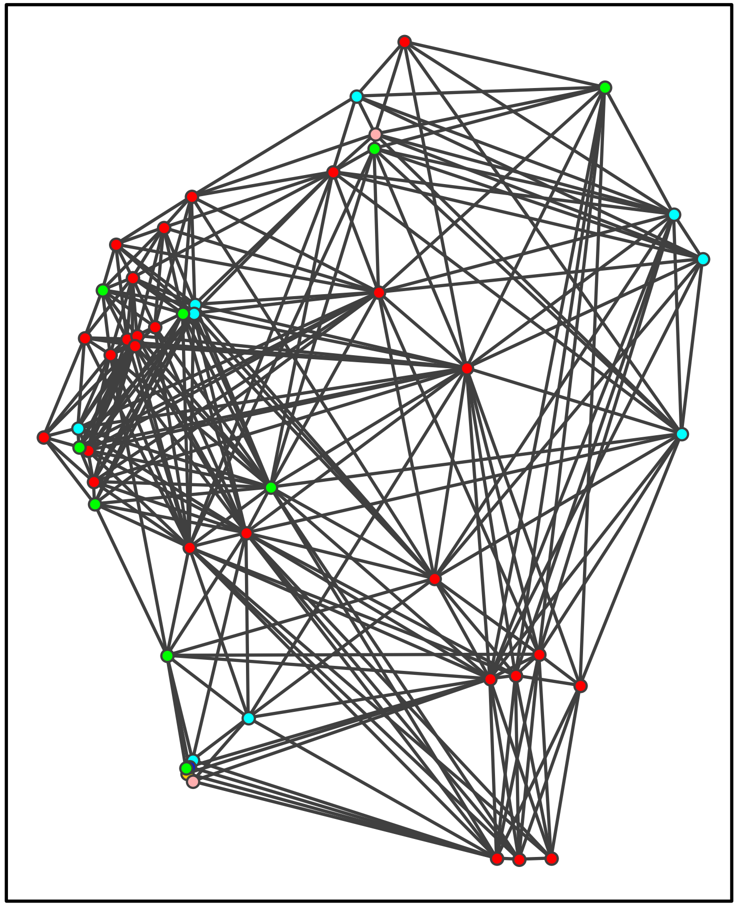


$\beta = 0.5$

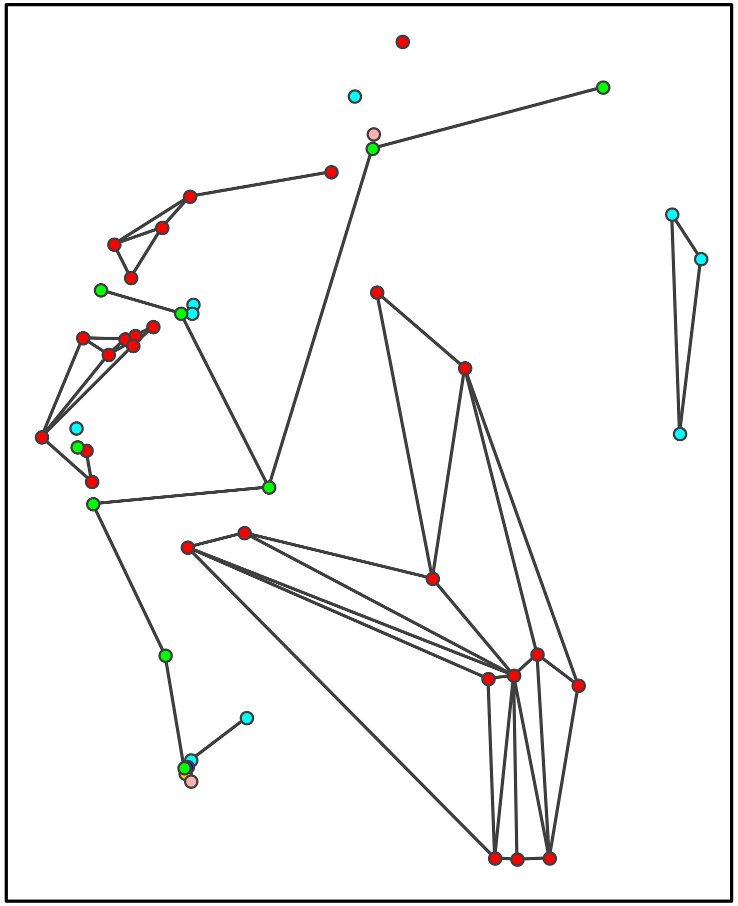


$\beta = 0.9$

Experiment setup

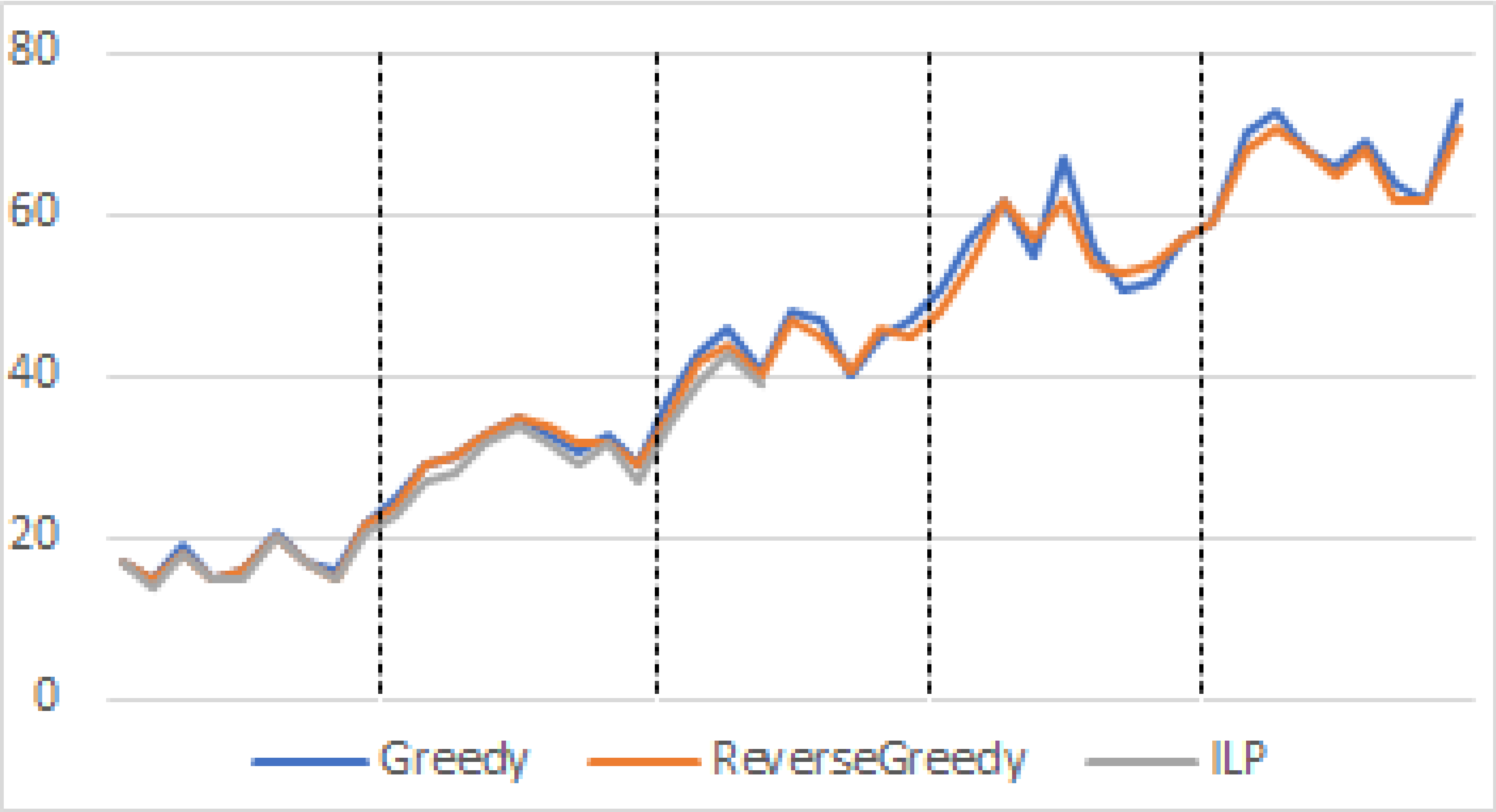


$\beta = 0.5$



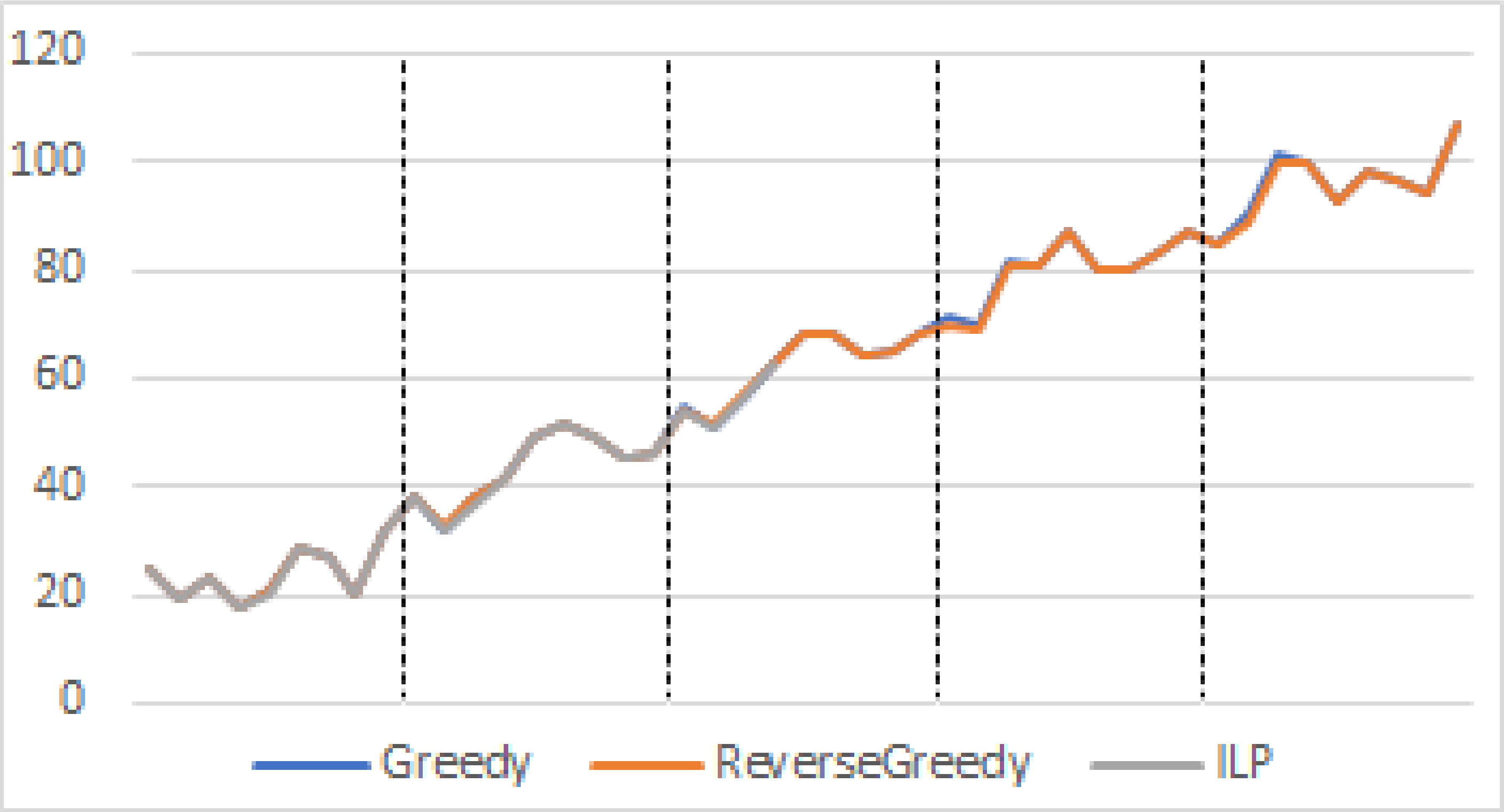
50 points, 15 clusters

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

Problem	Quality	Complexity
Cluster Min.	exact	NP-hard
– Greedy	no const. factor	$n + k + m \log m$
– Rev. Greedy	no const. factor	$n + k \log k + m \log m$
– 1-plane graphs	exact	$n \log n$
Edge Max.	exact	NP-hard

Summary and Future Work

Problem	Quality	Complexity
Cluster Min.	exact	NP-hard
– Greedy	no const. factor	$n + k + m \log m$
– Rev. Greedy	no const. factor	$n + k \log k + m \log m$
– 1-plane graphs	exact	$n \log n$
Edge Max.	exact	NP-hard

- 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

Problem	Quality	Complexity
Cluster Min.	exact	NP-hard
– Greedy	no const. factor	$n + k + m \log m$
– Rev. Greedy	no const. factor	$n + k \log k + m \log m$
– 1-plane graphs	exact	$n \log n$
Edge Max.	exact	NP-hard

- 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

Problem	Quality	Complexity
Cluster Min.	exact	NP-hard
– Greedy	no const. factor	$n + k + m \log m$
– Rev. Greedy	no const. factor	$n + k \log k + m \log m$
– 1-plane graphs	exact	$n \log n$
Edge Max.	exact	NP-hard

- 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?