EUROVIS ZURICH 21

ClusterSets: Optimizing Planar Clusters in Categorical Point Data

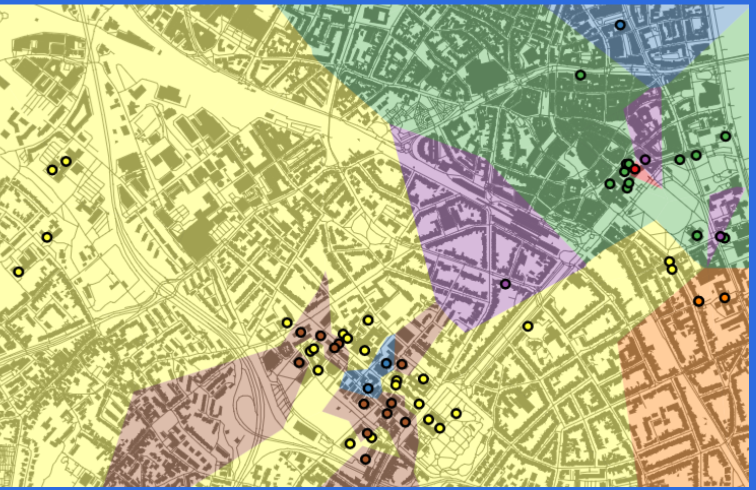
Jakob Geiger
Sabine Cornelsen
Jan-Henrik Haunert
Philipp Kindermann
Tamara Mchedlidze
Martin Nöllenburg
Yoshio Okamoto
Alexander Wolff

EUROVIS ZURICH 21

ClusterSets: Optimizing Planar Clusters in Categorical Point Data

Jakob Geiger
Jan-Henrik Haunert
Tamara Mchedlidze
Yoshio Okamoto
Sabine Cornelsen
Philipp Kindermann
Martin Nöllenburg
Alexander Wolff

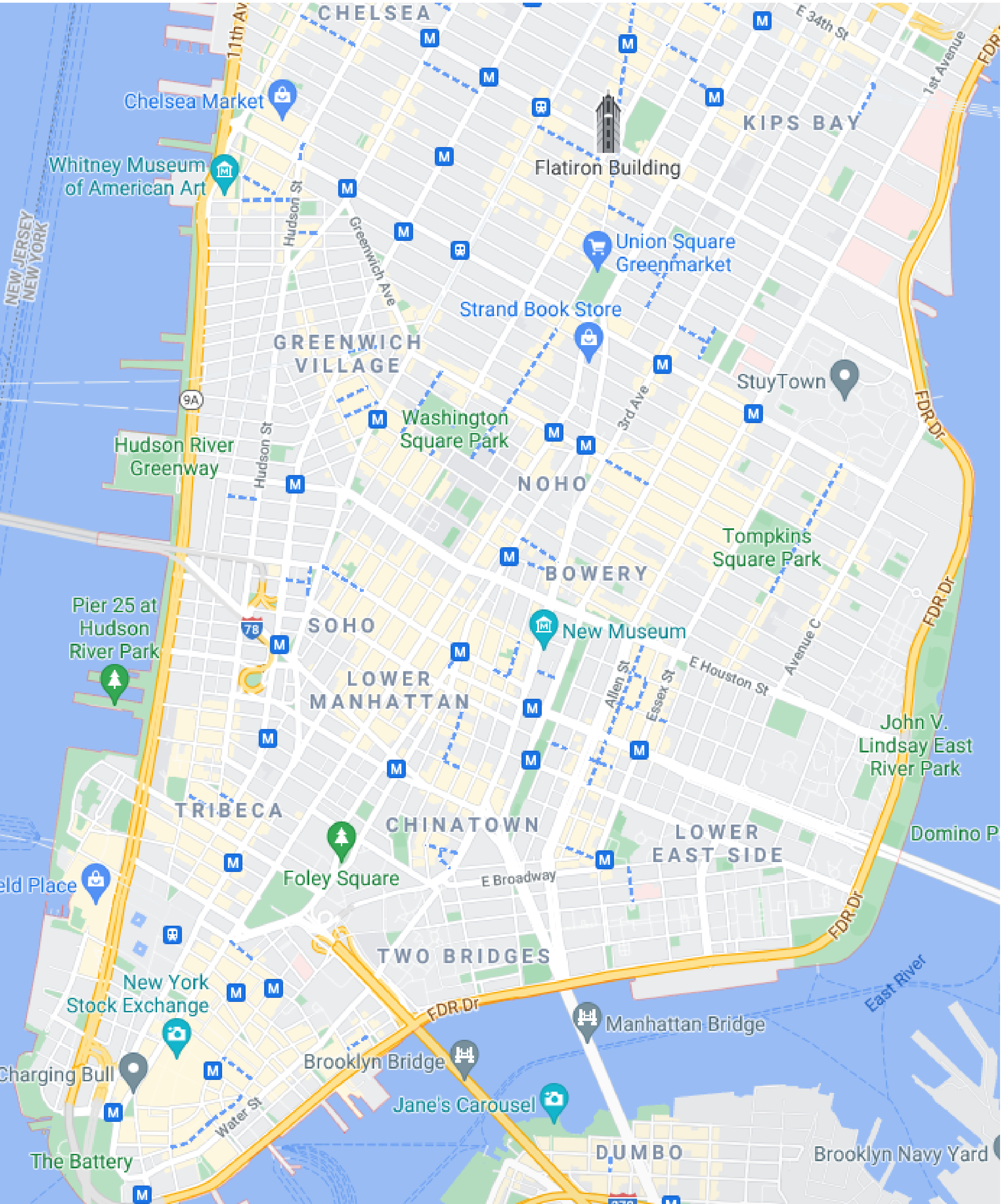
ClusterSets:
Optimizing Planar Clusters in Categorical Point Data



Jakob Geiger
Jan-Henrik Haunert
Tamara Mchedlidze
Yoshio Okamoto
Sabine Cornelsen
Philipp Kindermann
Martin Nöllenburg
Alexander Wolff

EUROVIS ZURICH 21

Sets of Points in Spatial Data

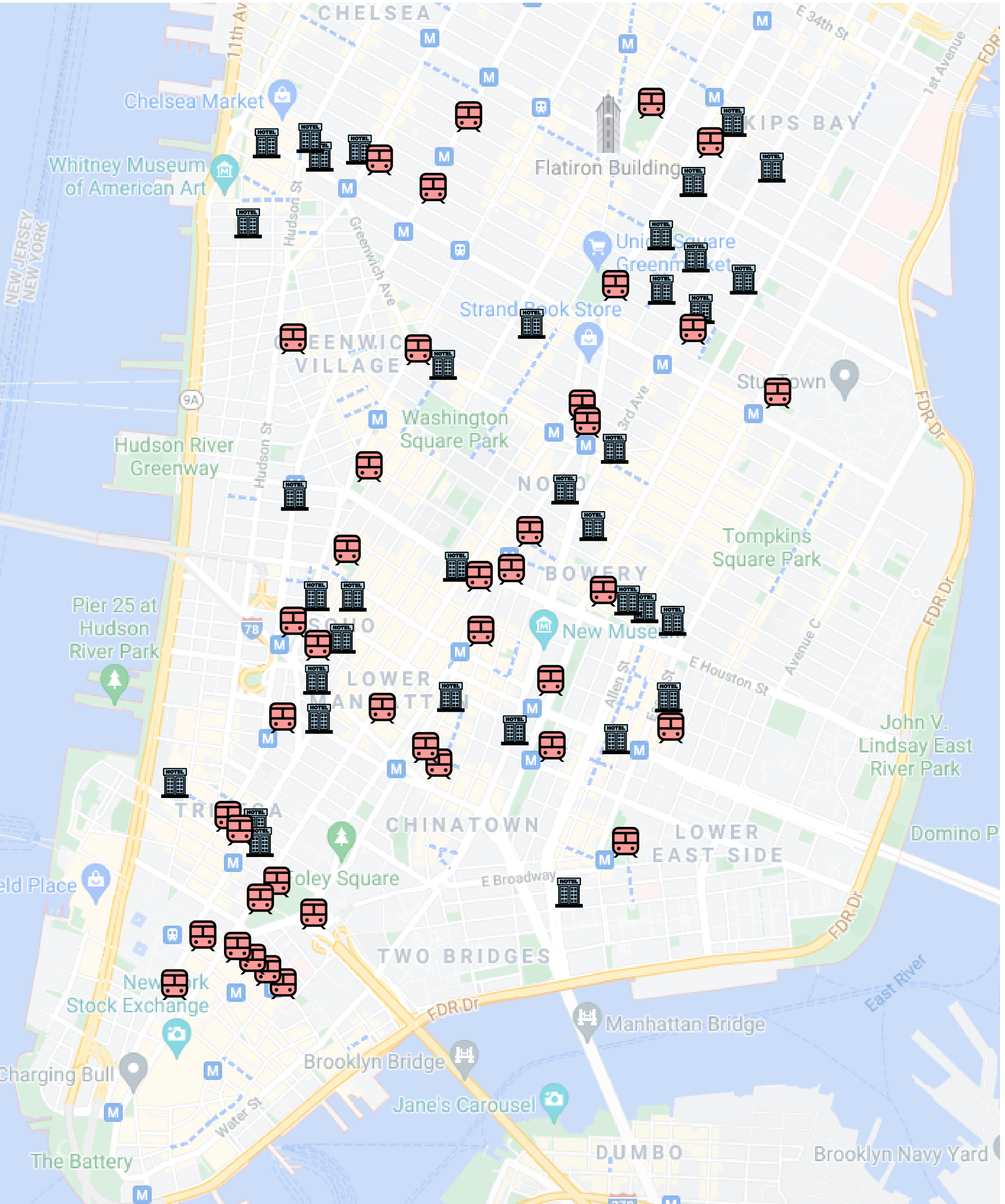


© Google Maps

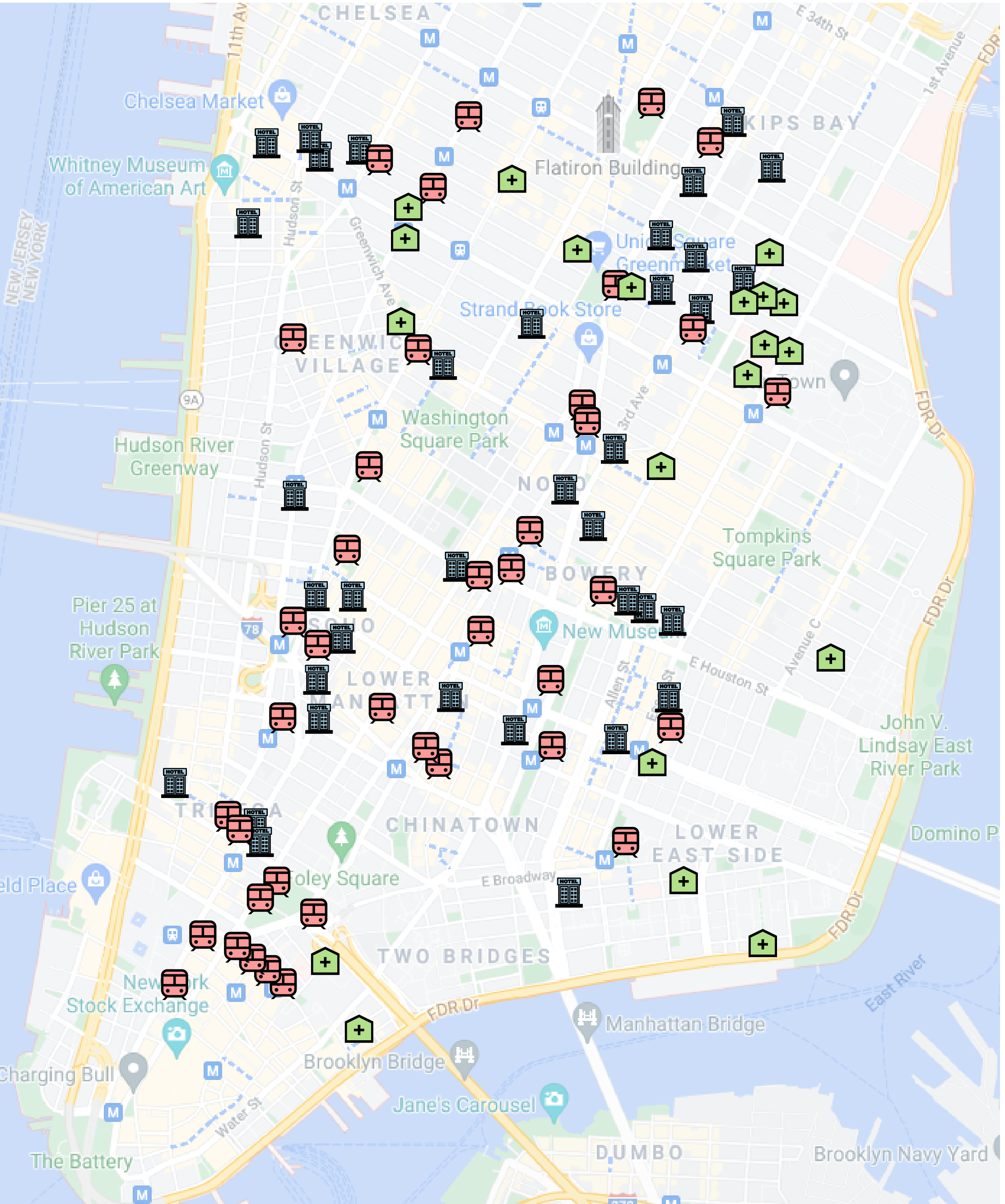
Sets of Points in Spatial Data

© Google Maps
Icons by icons8

Sets of Points in Spatial Data



Sets of Points in Spatial Data

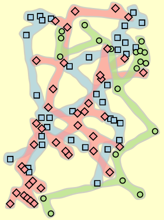


© Google Maps
Icons by icons8

Sets of Points in Spatial Data

Bubble Sets

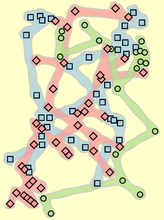
[Collins, Penn & Carpendal '09]



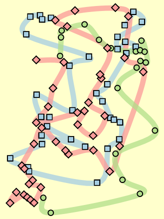
© Google Maps
Icons by icons8

Sets of Points in Spatial Data

Bubble Sets
[Collins, Penn & Carpendal '09]

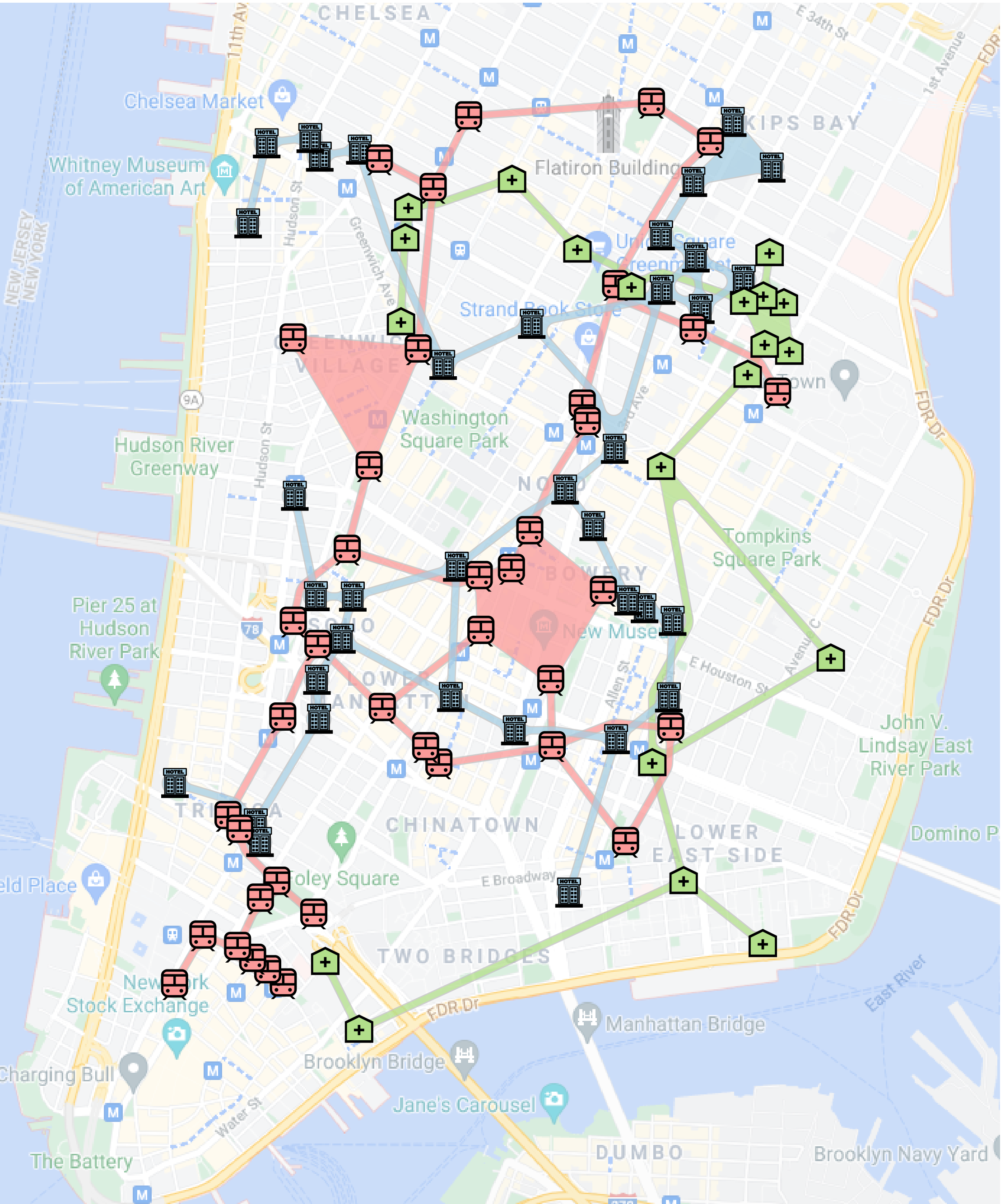


LineSets
[Alper, Henry Riche, Ramos & Czerwinski '11]

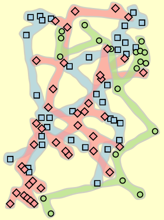


© Google Maps
Icons by icons8

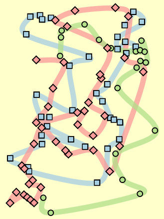
Sets of Points in Spatial Data



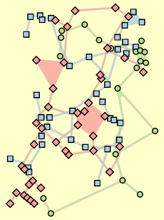
Bubble Sets
[Collins, Penn & Carpendal '09]



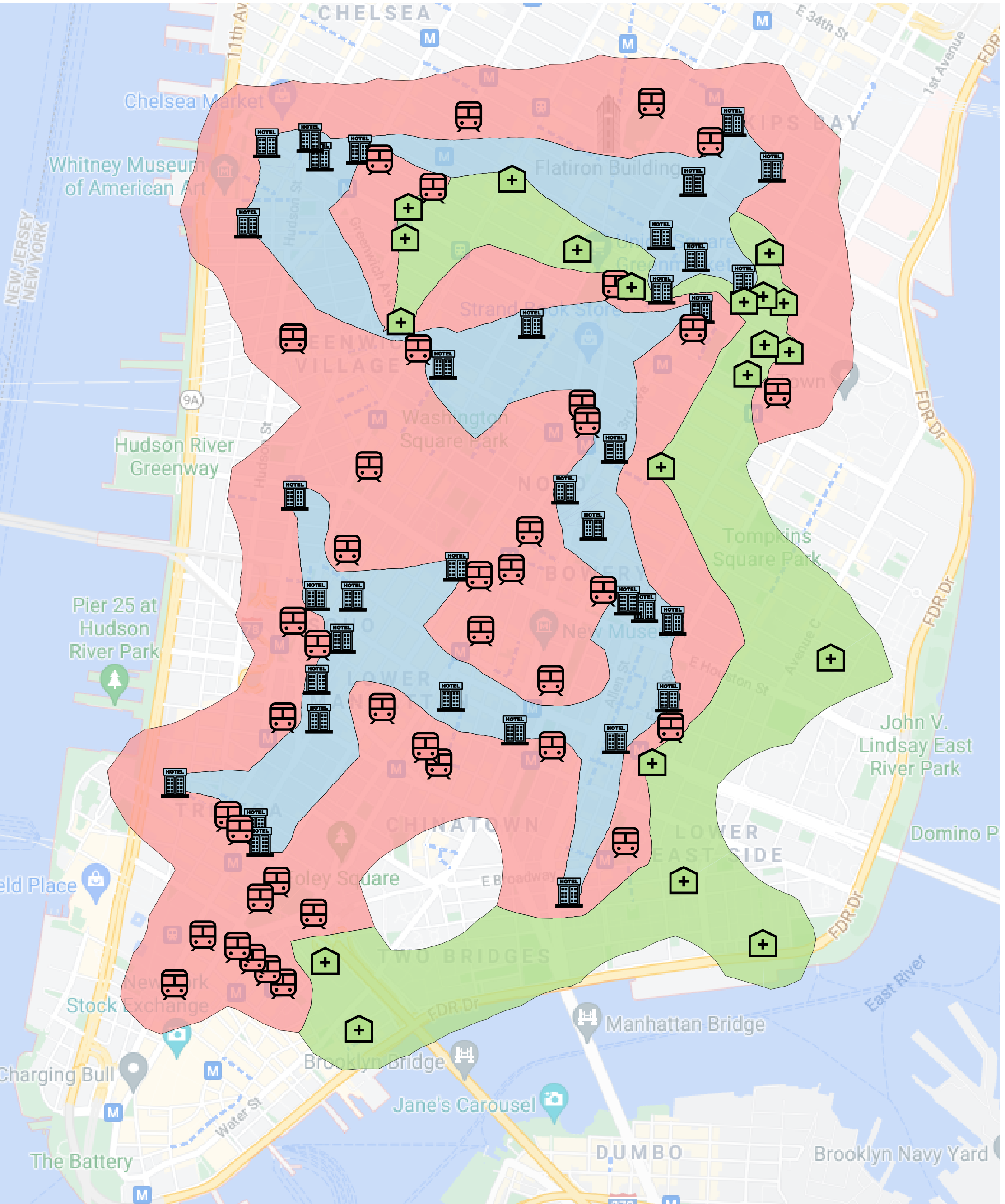
LineSets
[Alper, Henry Riche, Ramos & Czerwinski '11]



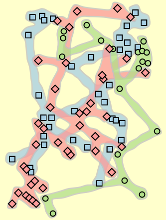
KelpFusion
[Meulemans, Henry Riche, Speckmann, Alper & Dwyer '13]



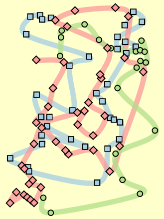
Sets of Points in Spatial Data



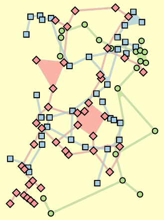
Bubble Sets
[Collins, Penn & Carpendal '09]



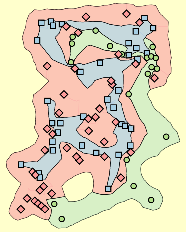
LineSets
[Alper, Henry Riche, Ramos & Czerwinski '11]



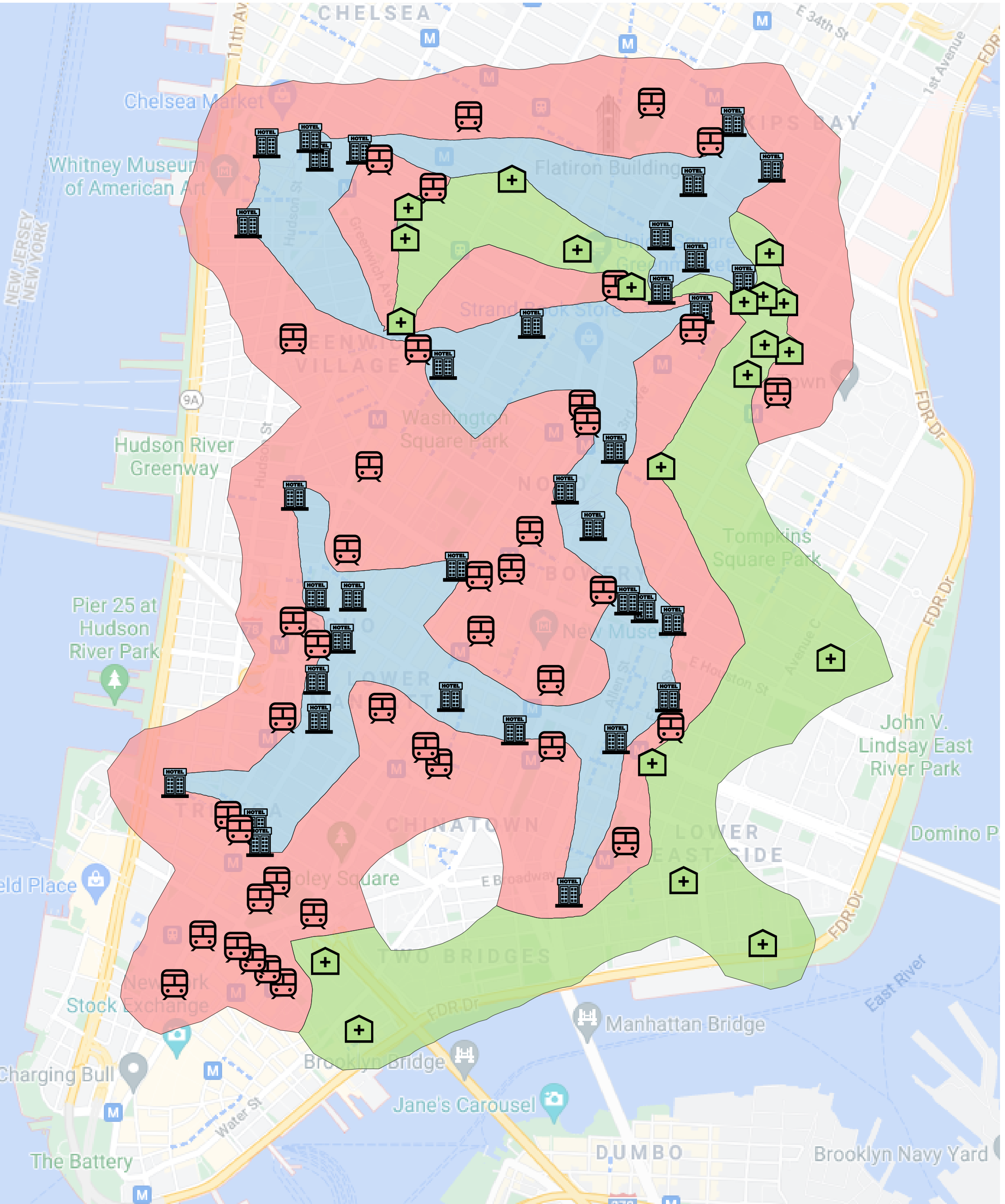
KelpFusion
[Meulemans, Henry Riche, Speckmann, Alper & Dwyer '13]



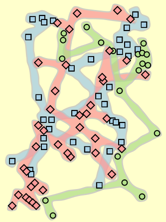
MapSets
[Efrat, Hu, Kobourov & Pupyrev '15]



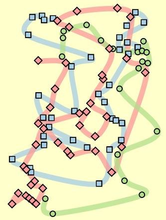
Sets of Points in Spatial Data



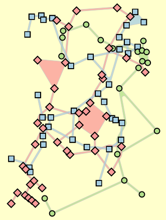
Bubble Sets
[Collins, Penn & Carpendal '09]



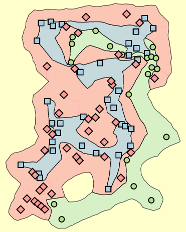
LineSets
[Alper, Henry Riche, Ramos & Czerwinski '11]



KelpFusion
[Meulemans, Henry Riche, Speckmann, Alper & Dwyer '13]

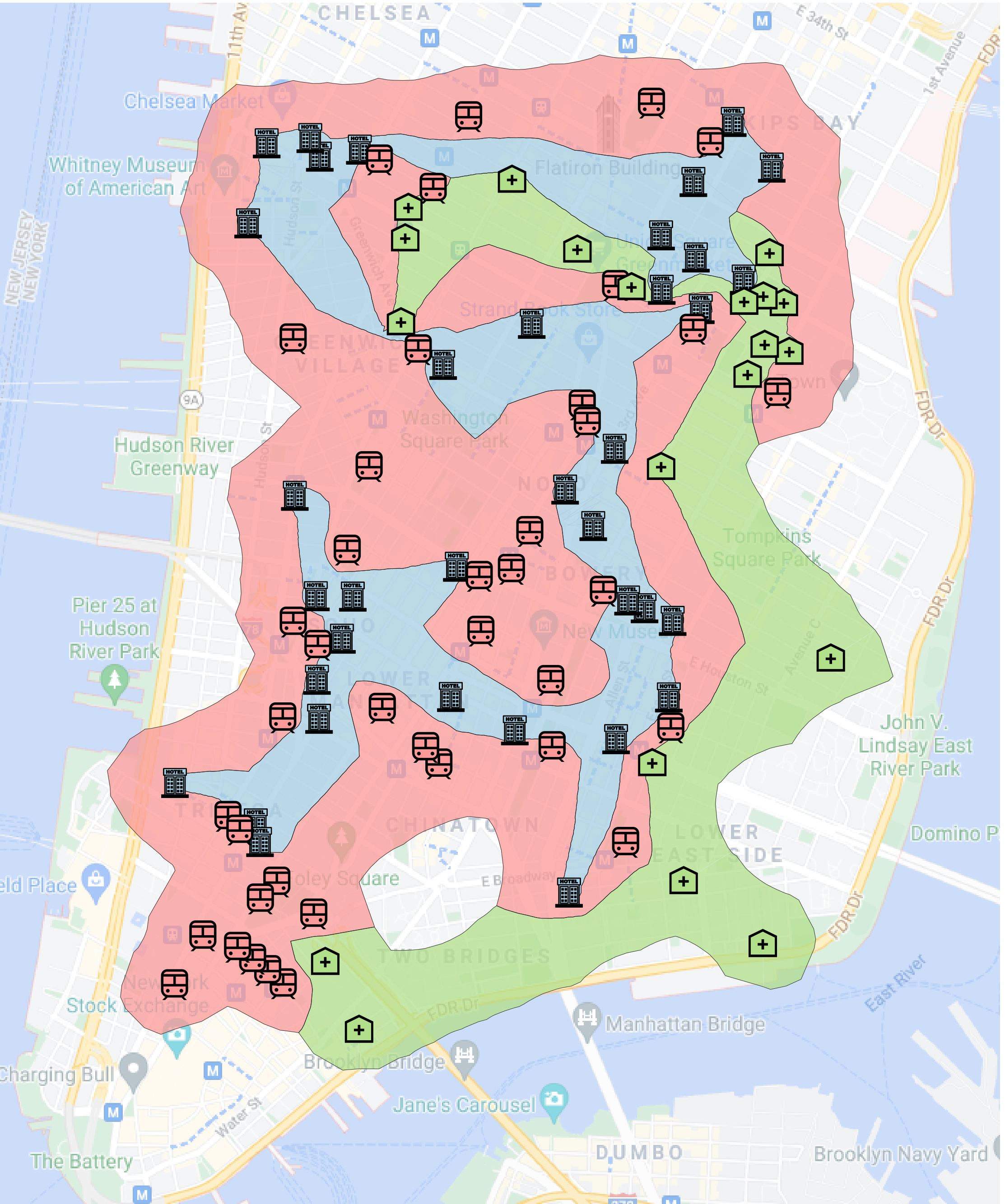


MapSets
[Efrat, Hu, Kobourov & Pupyrev '15]

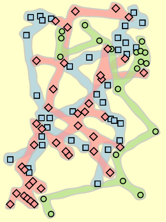


All points of the same category are connected.

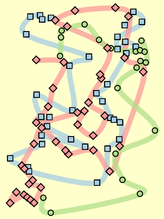
Sets of Points in Spatial Data



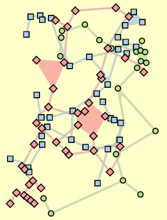
Bubble Sets
[Collins, Penn & Carpendal '09]



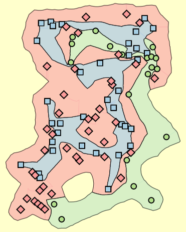
LineSets
[Alper, Henry Riche, Ramos & Czerwinski '11]



KelpFusion
[Meulemans, Henry Riche, Speckmann, Alper & Dwyer '13]



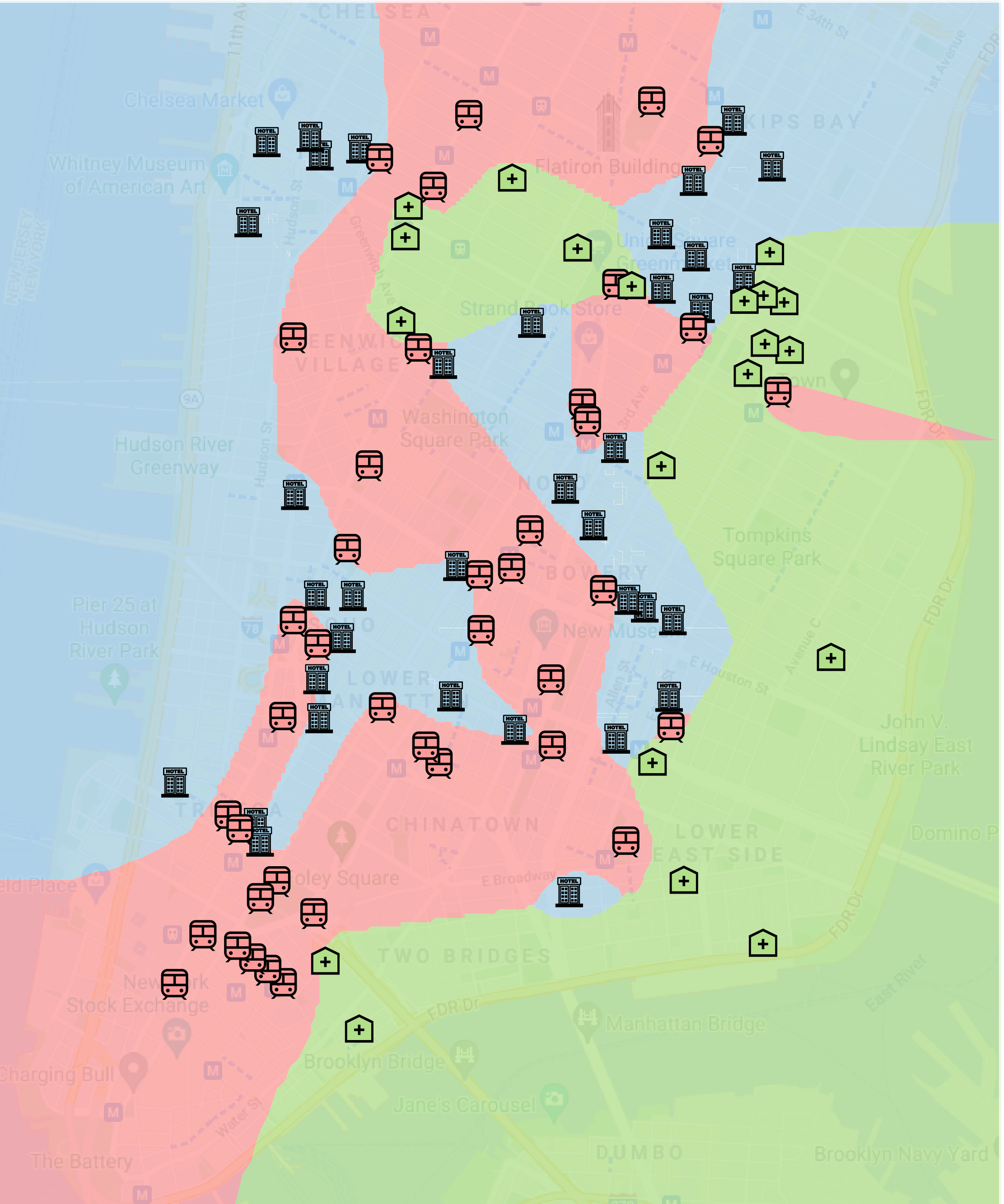
MapSets
[Efrat, Hu, Kobourov & Pupyrev '15]



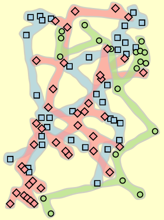
All points of the same category are connected.

Relax connectivity requirement
→ preservation of locality of clusters

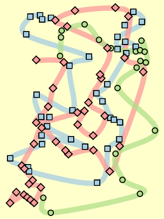
Sets of Points in Spatial Data



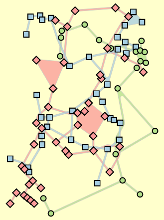
Bubble Sets
[Collins, Penn & Carpendal '09]



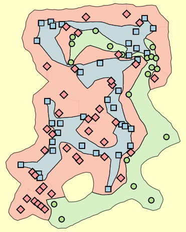
LineSets
[Alper, Henry Riche, Ramos & Czerwinski '11]



KelpFusion
[Meulemans, Henry Riche, Speckmann, Alper & Dwyer '13]



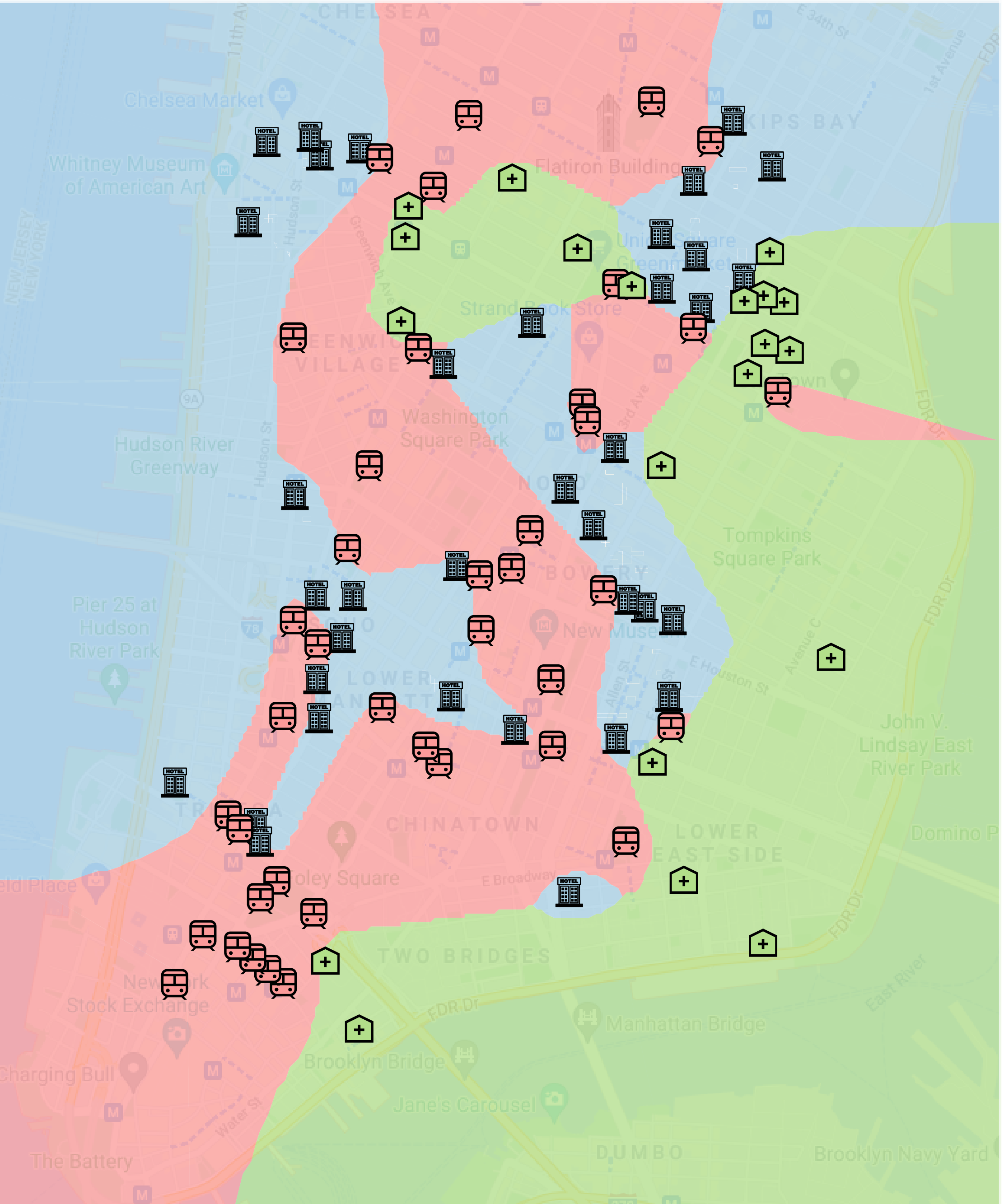
MapSets
[Efrat, Hu, Kobourov & Pupyrev '15]



All points of the same category are connected.

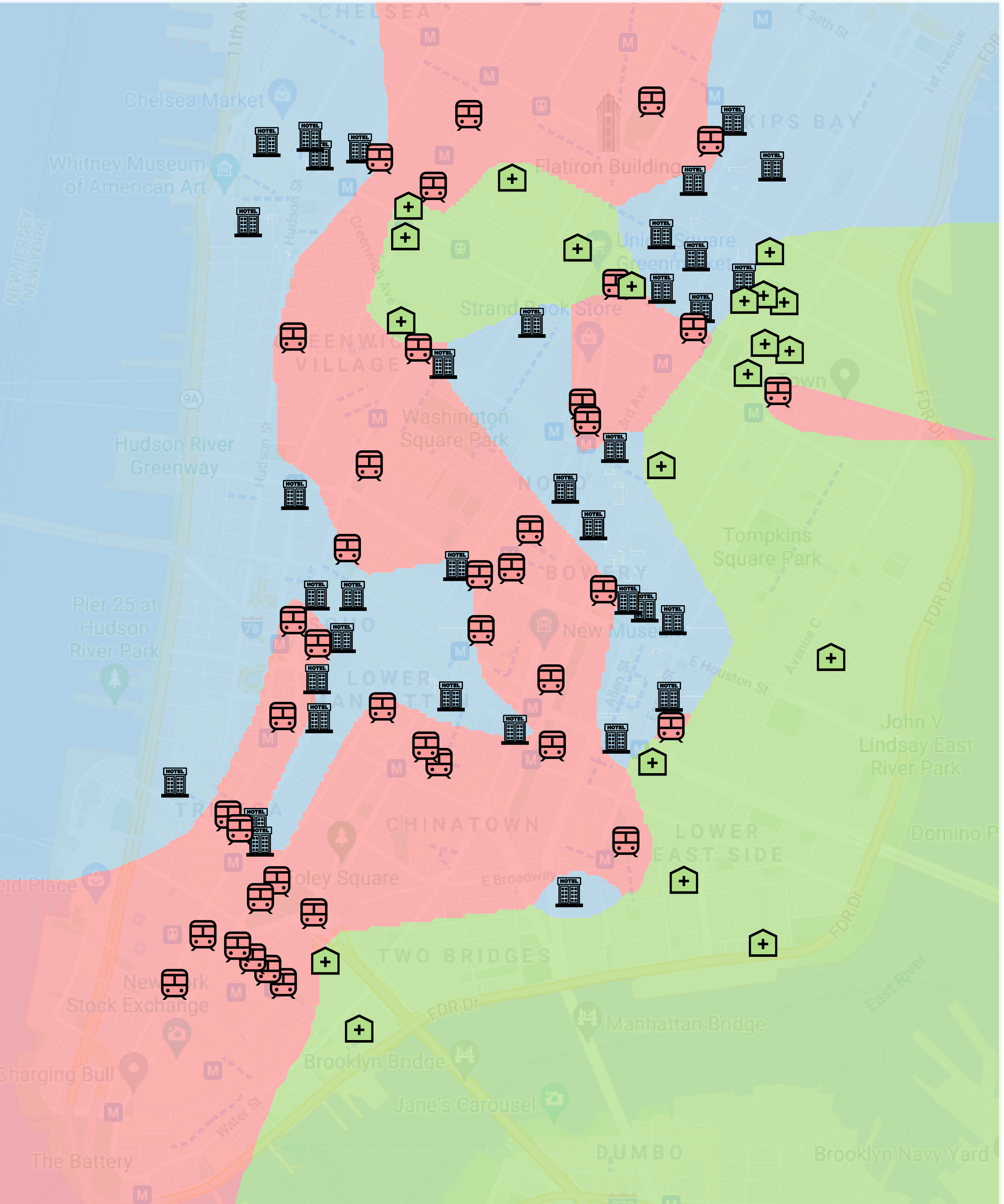
Relax connectivity requirement
→ preservation of locality of clusters

Design Goals



Relax connectivity requirement
→ preservation of locality of clusters

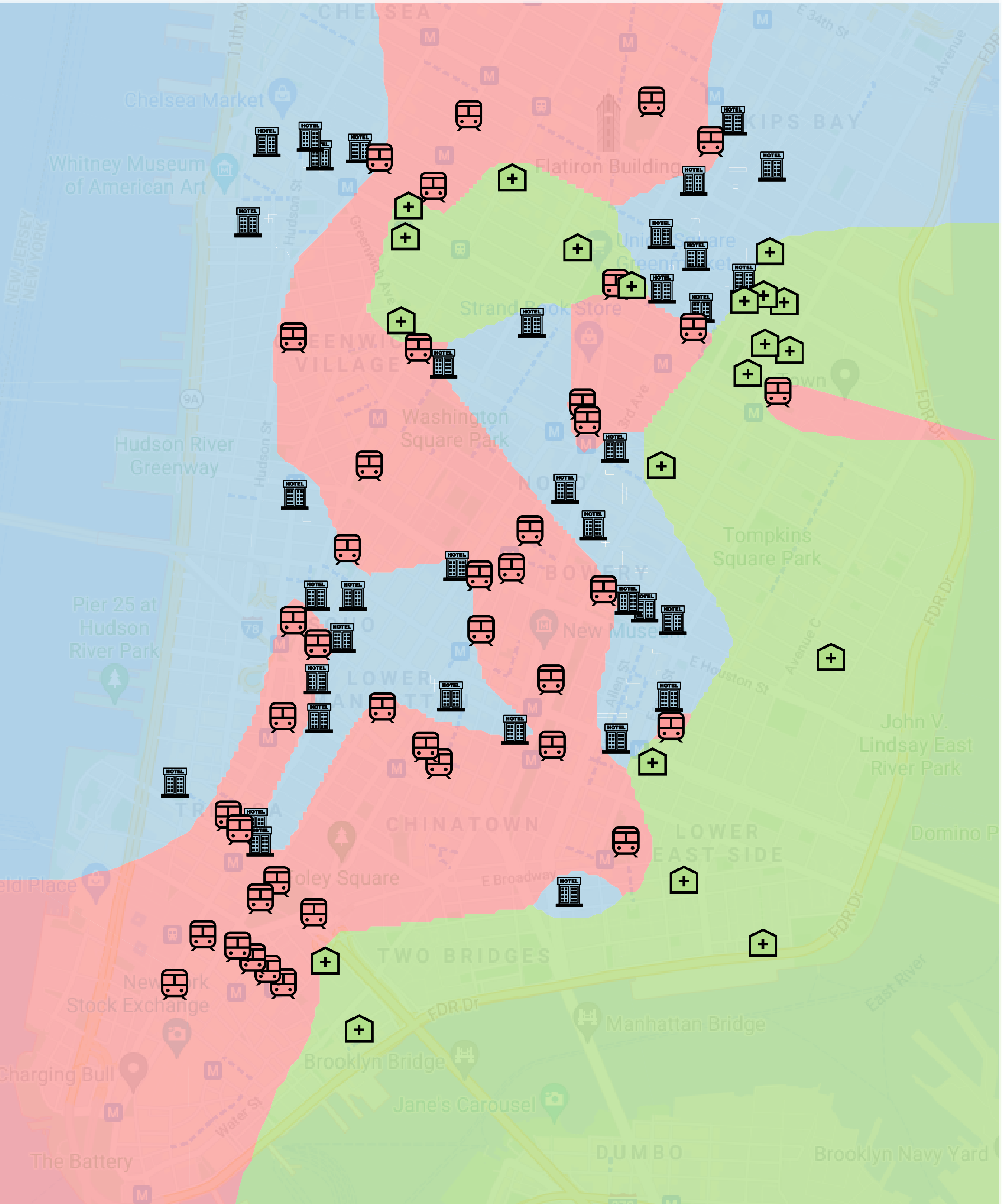
Design Goals



Relax connectivity requirement
→ preservation of locality of clusters

- Categories represented by distinct colors

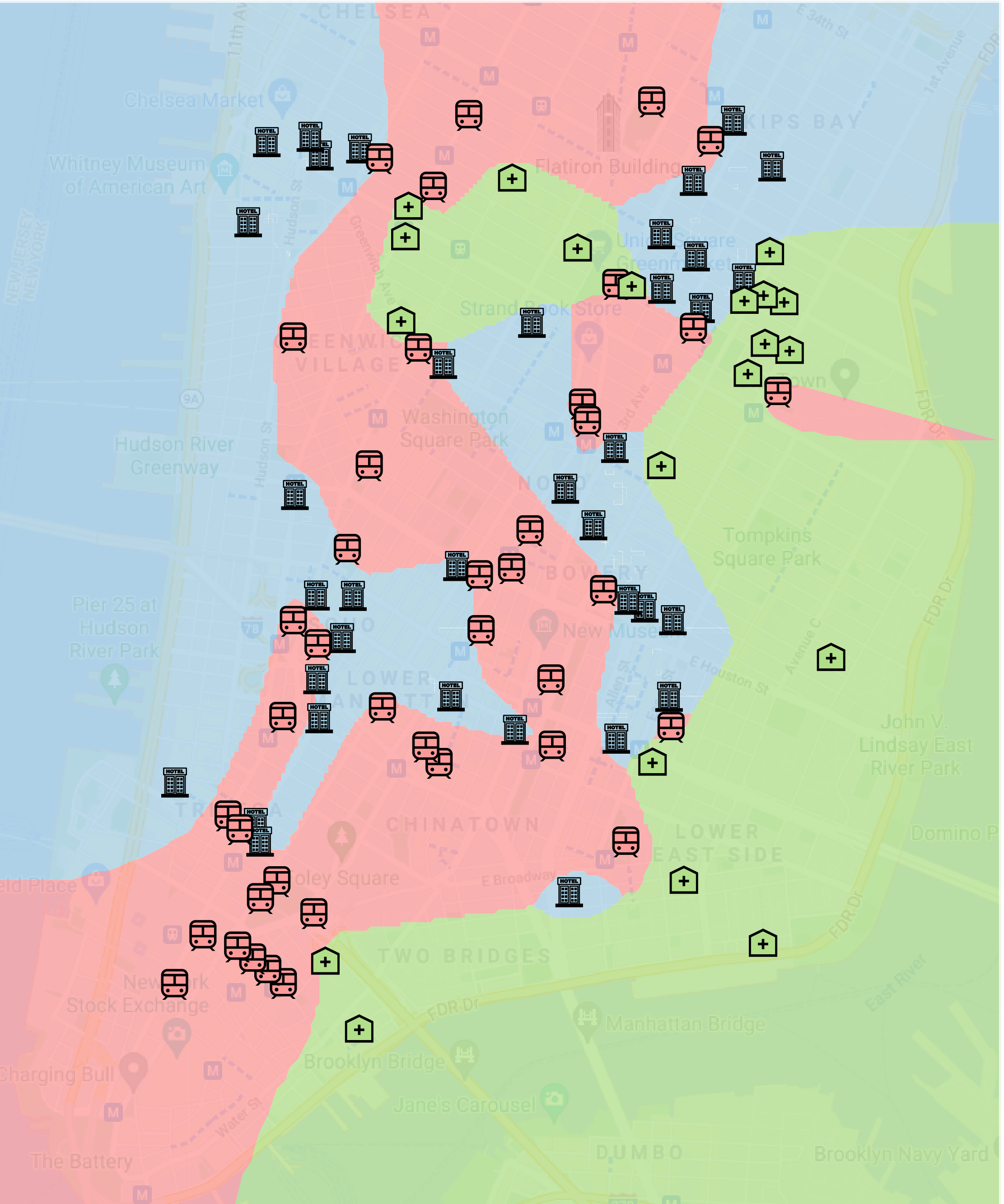
Design Goals



Relax connectivity requirement
→ preservation of locality of clusters

- Categories represented by distinct colors
- Clusters: Subset of points from same category

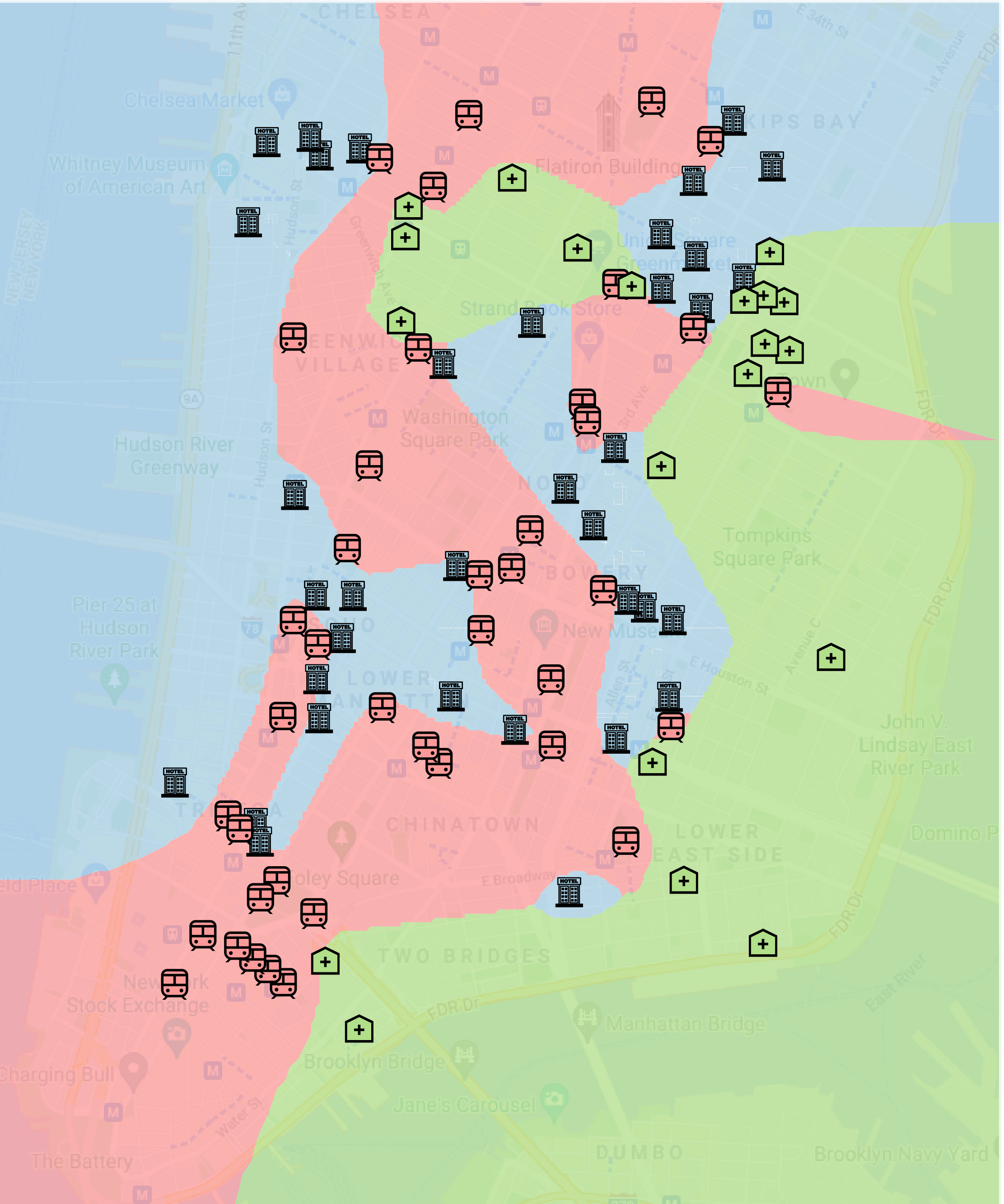
Design Goals



Relax connectivity requirement
→ preservation of locality of clusters

- Categories represented by distinct colors
- Clusters: Subset of points from same category
- Clusters form distinct regions

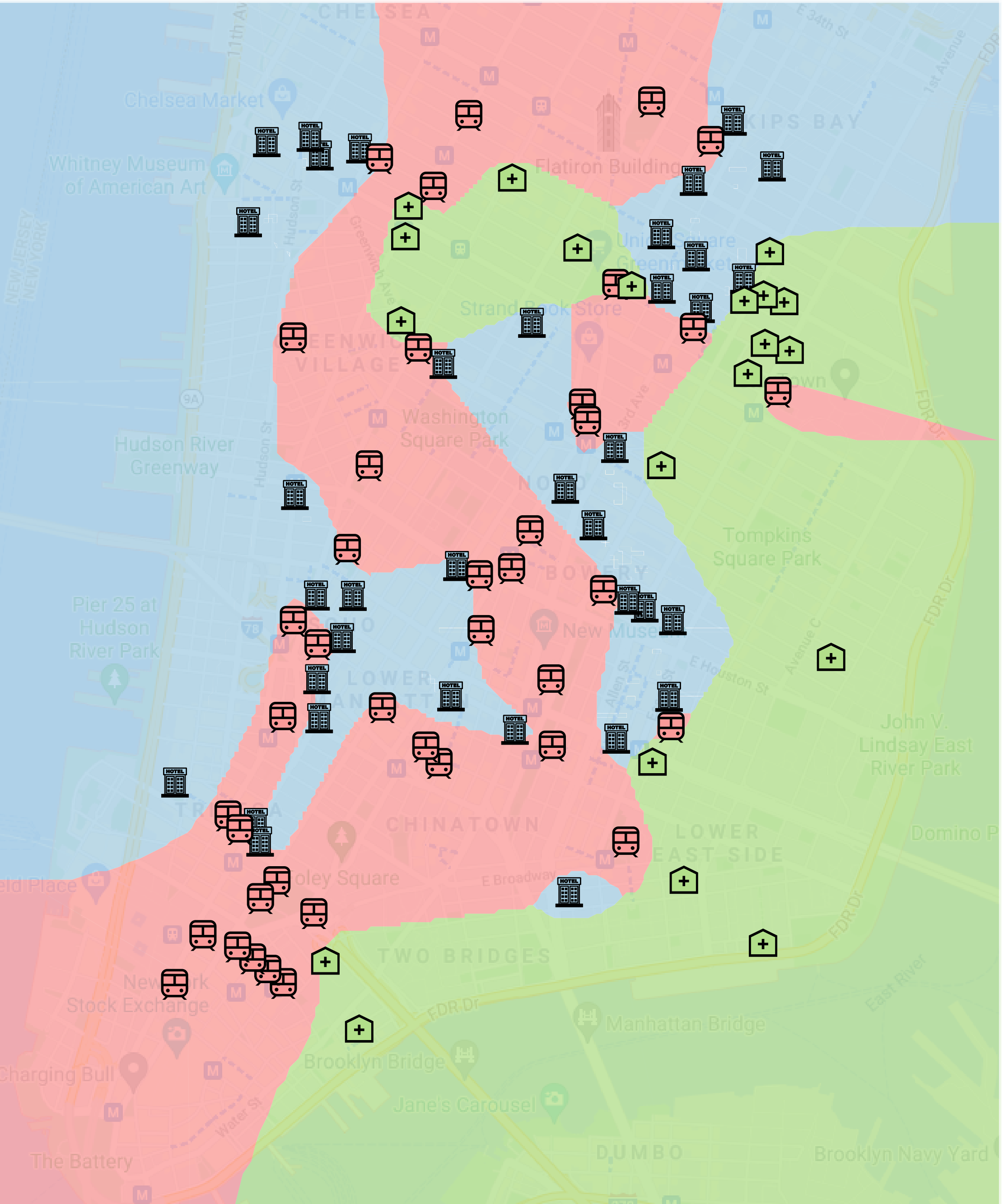
Design Goals



Relax connectivity requirement
→ preservation of locality of clusters

- Categories represented by distinct colors
- Clusters: Subset of points from same category
- Clusters form distinct regions
- Points in cluster are sufficiently close

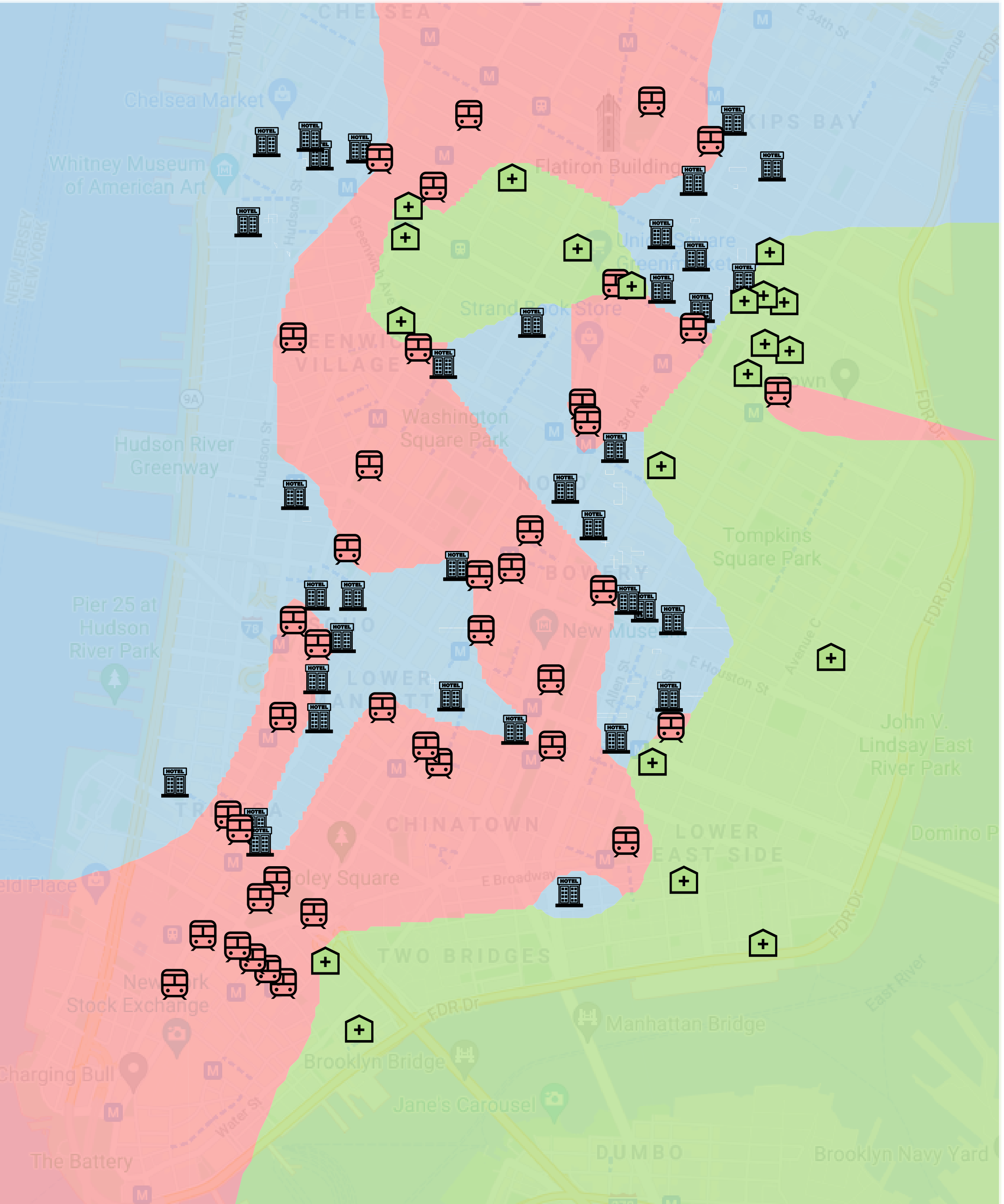
Design Goals



Relax connectivity requirement
$\rightarrow$ preservation of locality of clusters

- Categories represented by distinct colors
- Clusters: Subset of points from same category
- Clusters form distinct regions
- Points in cluster are sufficiently close
- Small number of clusters per category

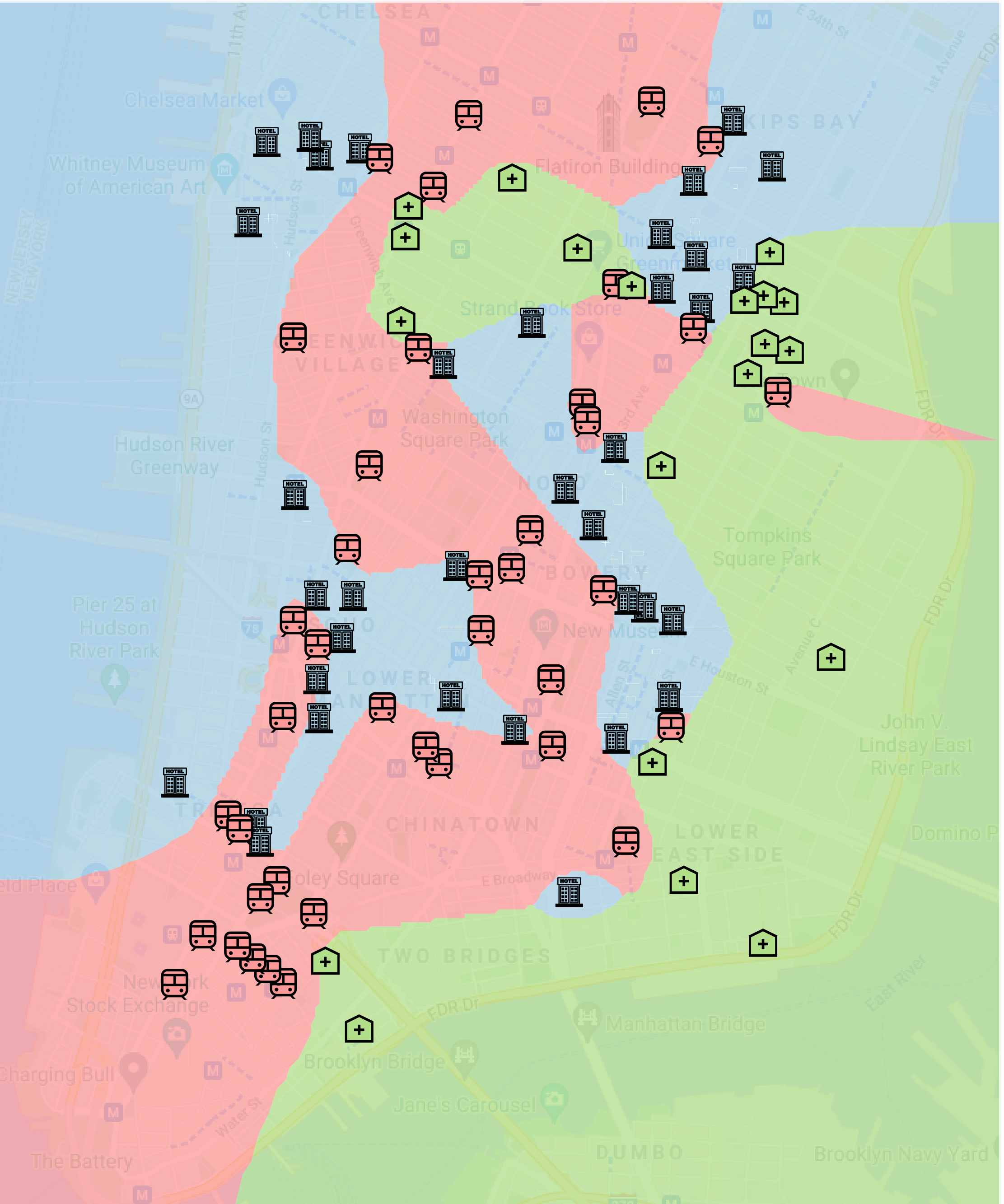
Design Goals



Relax connectivity requirement
→ preservation of locality of clusters

- Categories represented by distinct colors
- Clusters: Subset of points from same category
- Clusters form distinct regions
- Points in cluster are sufficiently close
- Small number of clusters per category

Design Goals

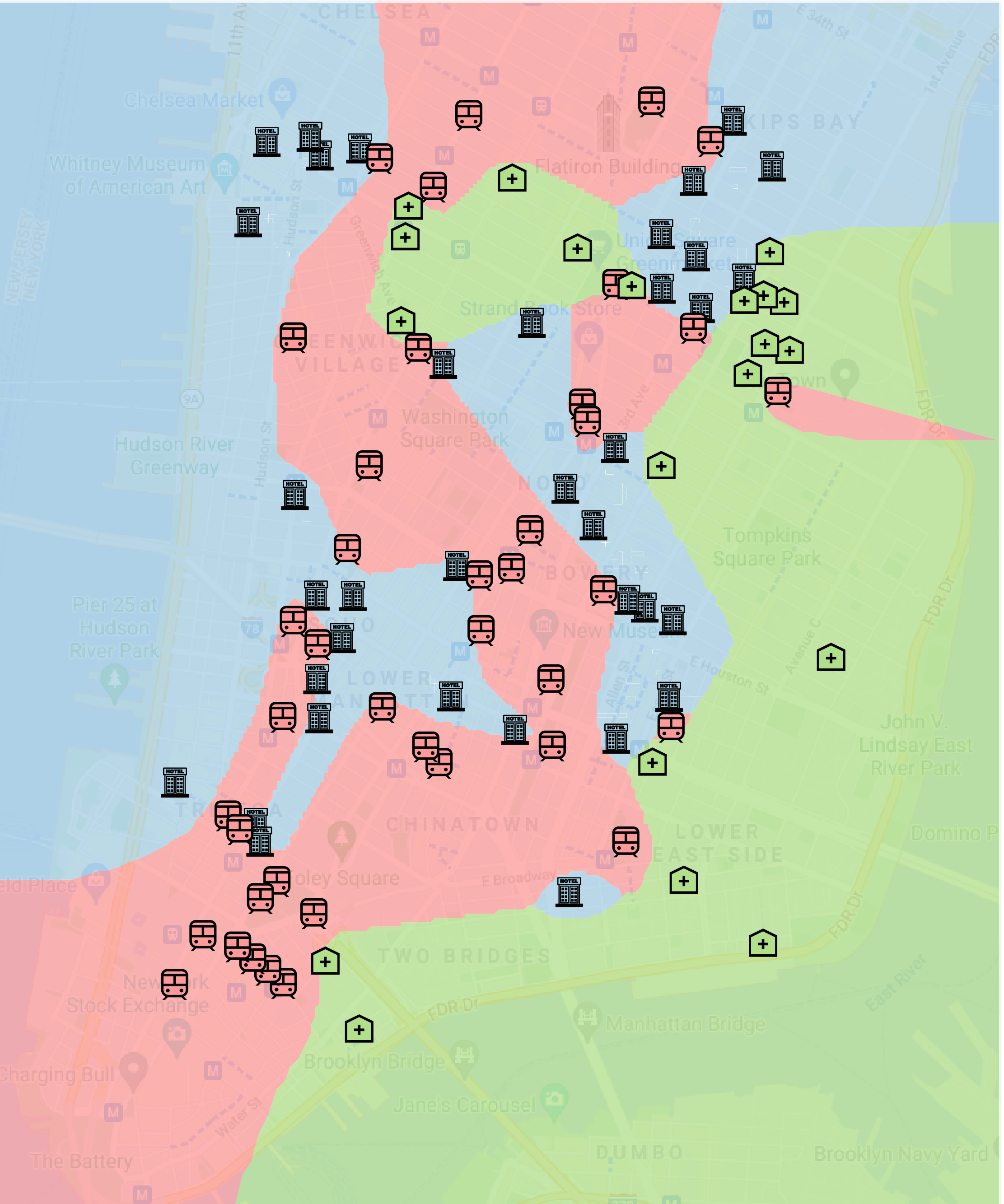


Relax connectivity requirement
$\rightarrow$ preservation of locality of clusters

- Categories represented by distinct colors
- Clusters: Subset of points from same category
- Clusters form distinct regions
- Points in cluster are sufficiently close
- Small number of clusters per category

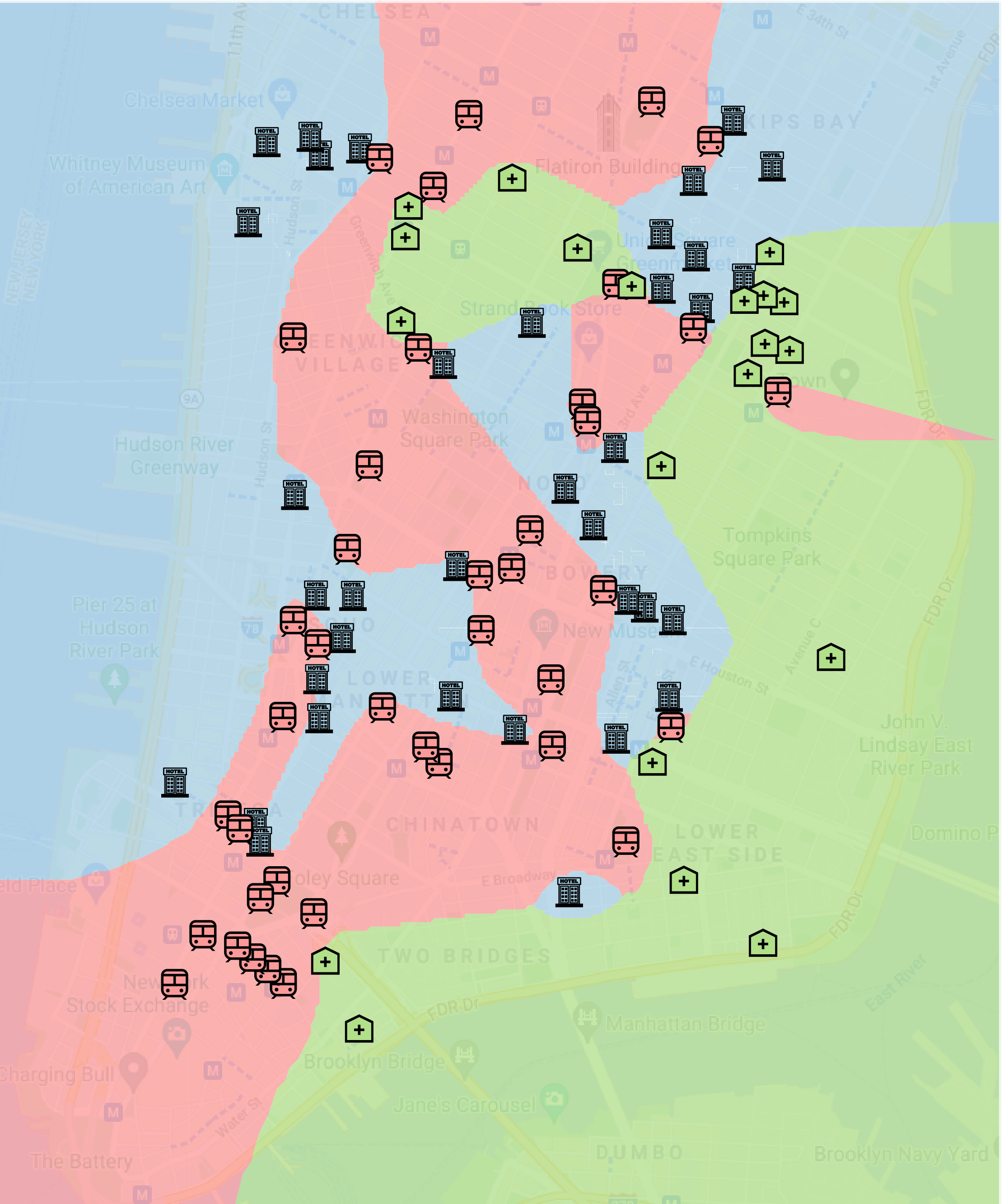
Points in same cluster connected
in a suitable proximity graph

Pipeline



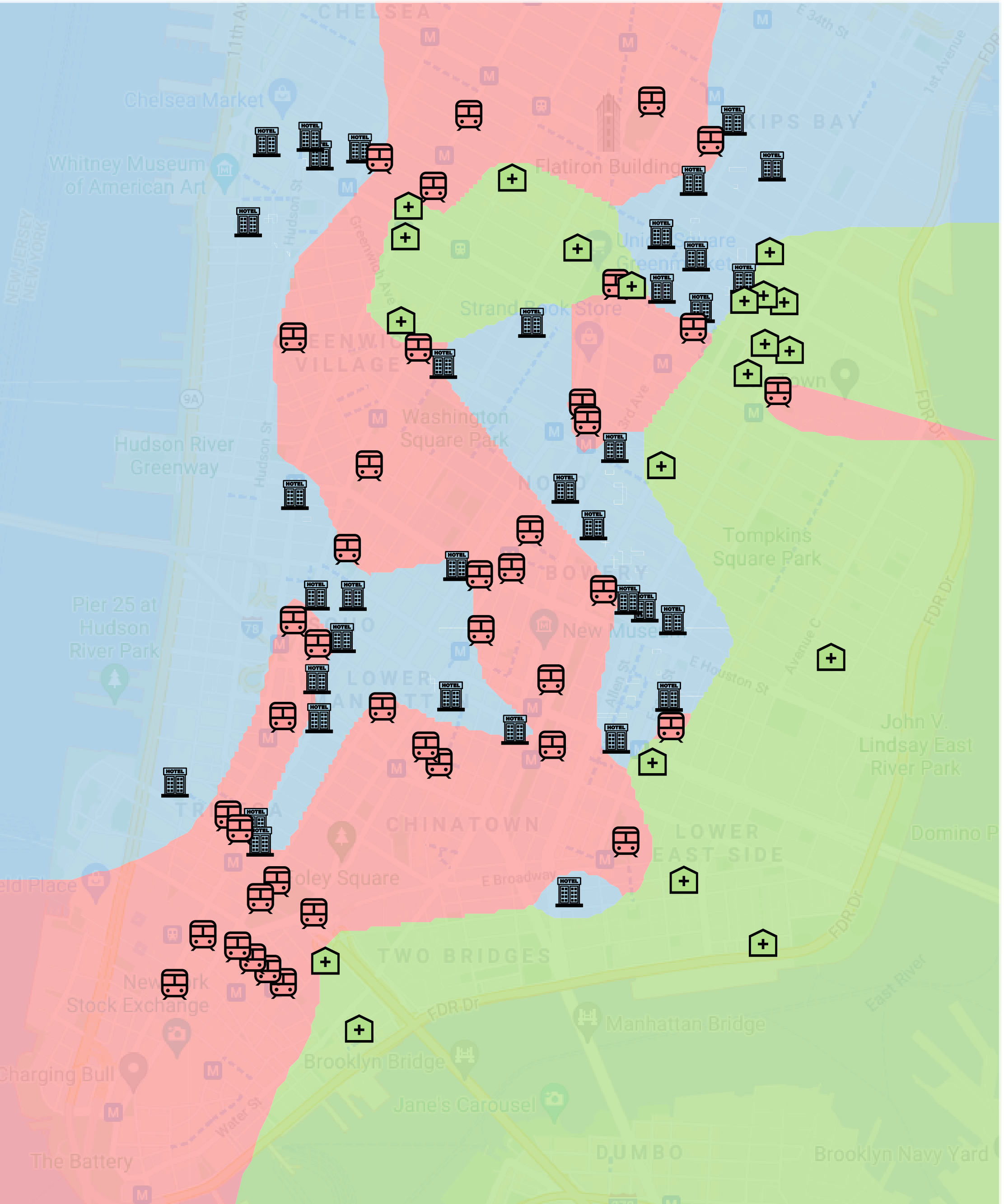
CHELSEA
Chelsea Market
Whitney Museum of American Art
Flatiron Building
KIPS BAY
Strand Book Store
GREENWICH VILLAGE
Washington Square Park
Hudson River Greenway
Tompkins Square Park
BOWERY
Pier 25 at Hudson River Park
New Museum
LOWER MANHATTAN
John V. Lindsay East River Park
CHINATOWN
LOWER EAST SIDE
Foley Square
TWO BRIDGES
New York Stock Exchange
Manhattan Bridge
Brooklyn Bridge
Charging Bull
Jane's Carousel
DUMBO
The Battery
Brooklyn Navy Yard

Pipeline

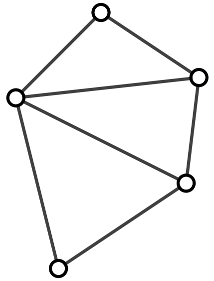


1. **Proximity Graph**

Pipeline

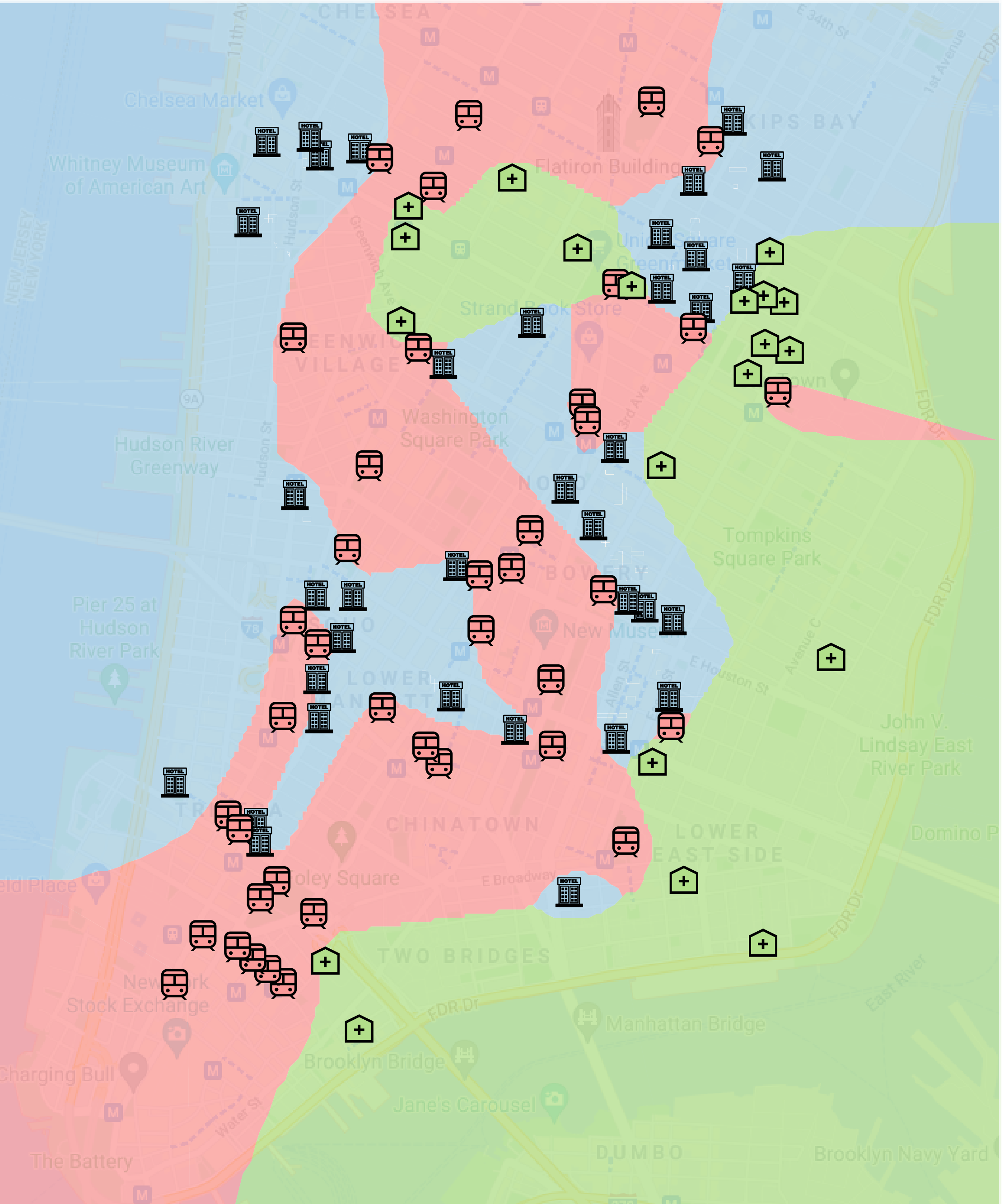


1. **Proximity Graph**

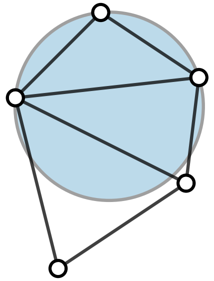


Delaunay Triangulation

Pipeline

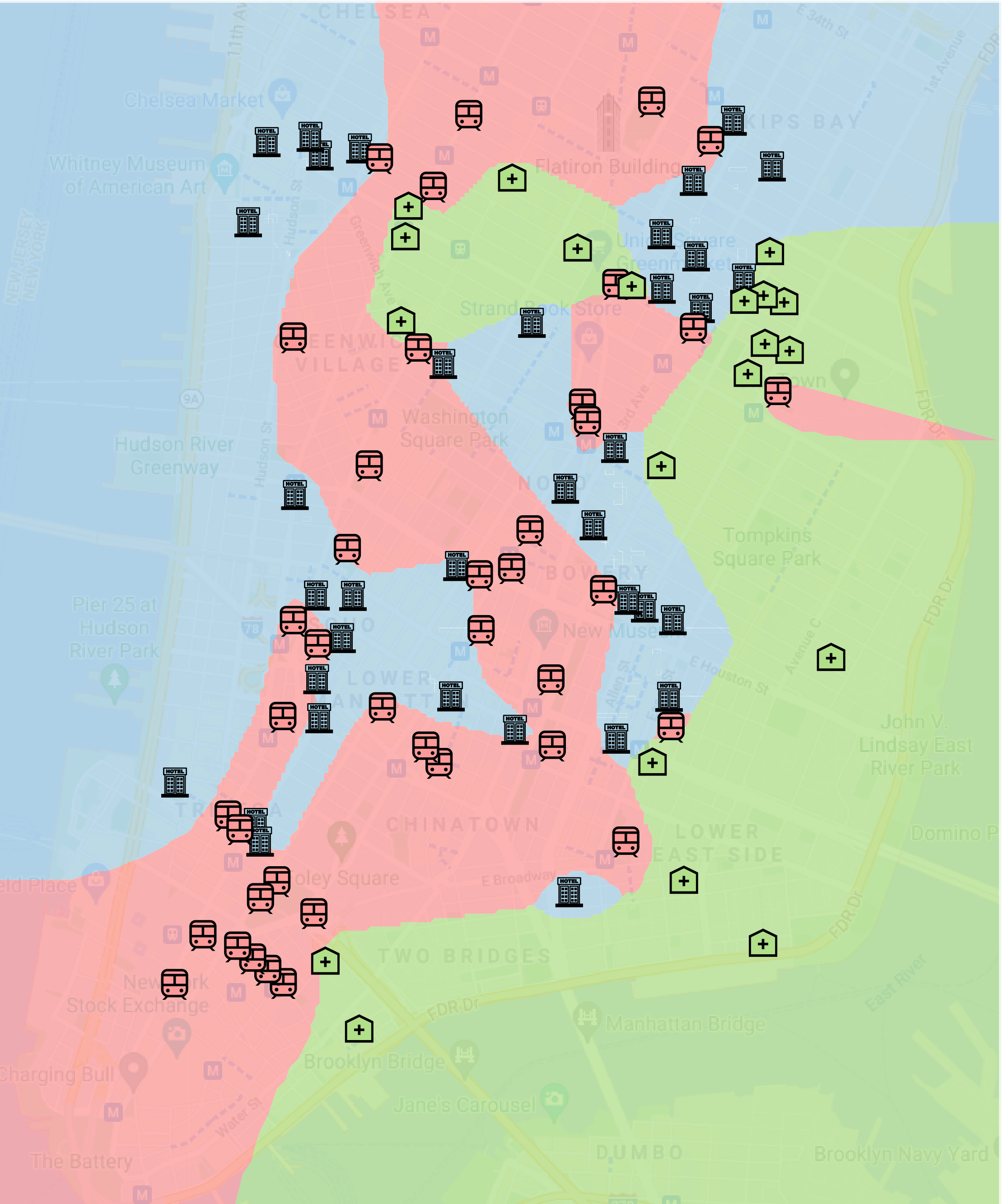


1. **Proximity Graph**

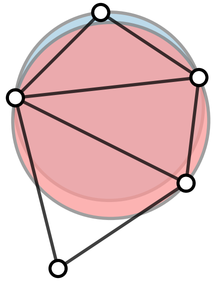


Delaunay Triangulation

Pipeline

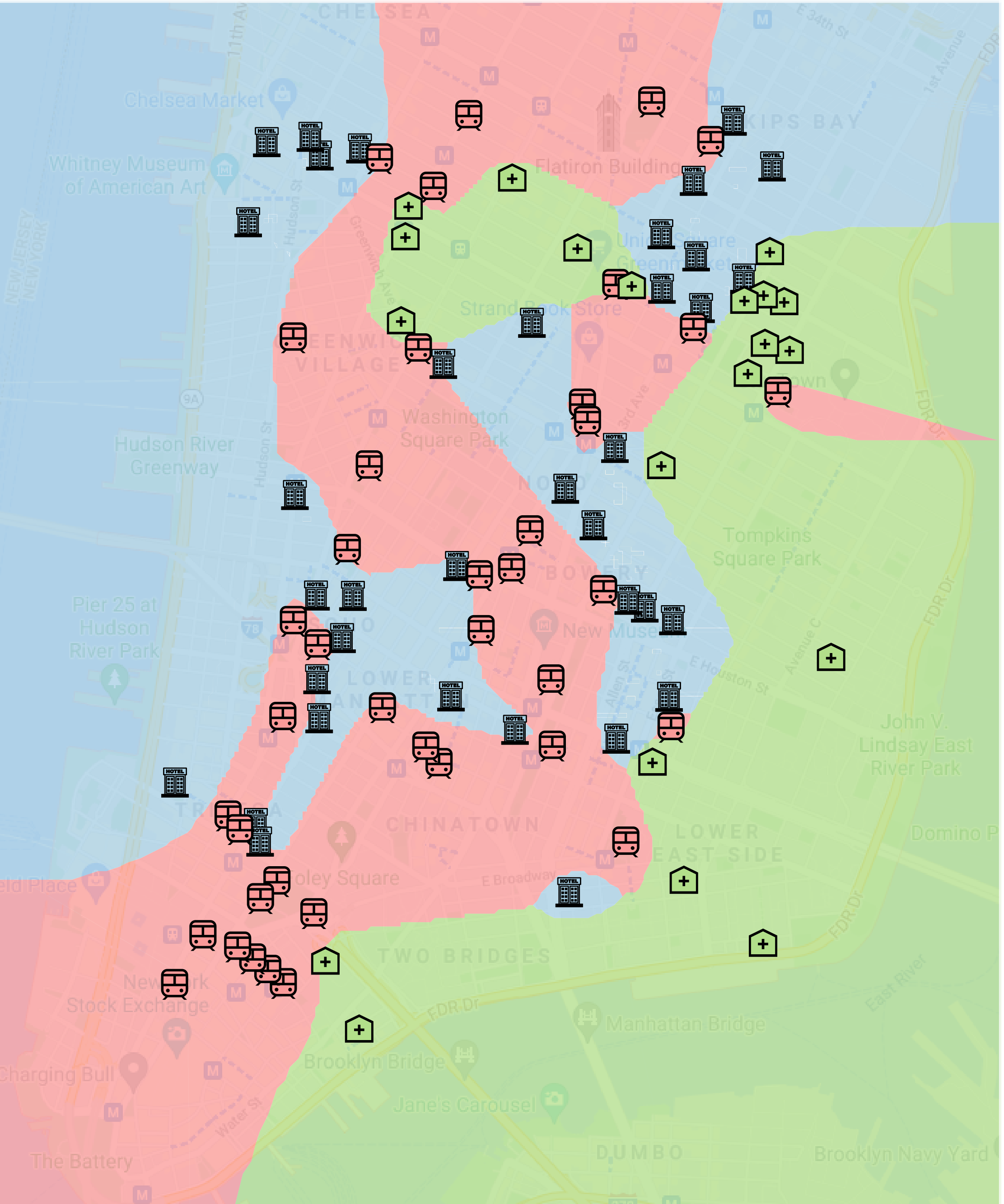


1. **Proximity Graph**

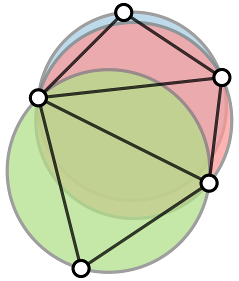


Delaunay Triangulation

Pipeline

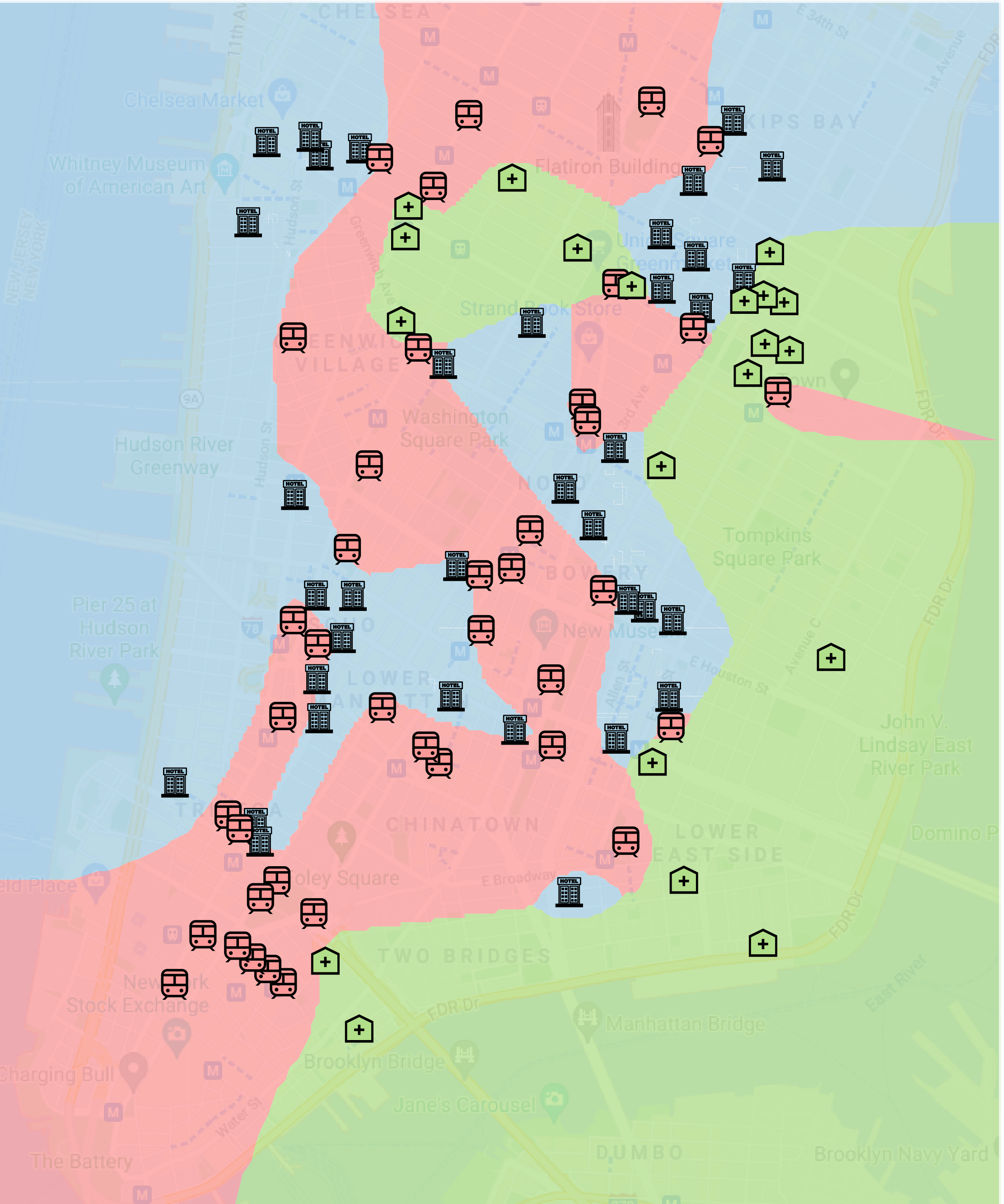


1. **Proximity Graph**

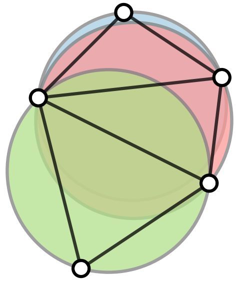


Delaunay Triangulation

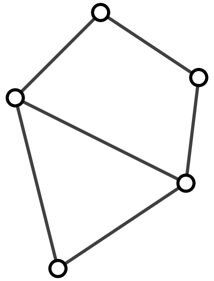
Pipeline



1. **Proximity Graph**

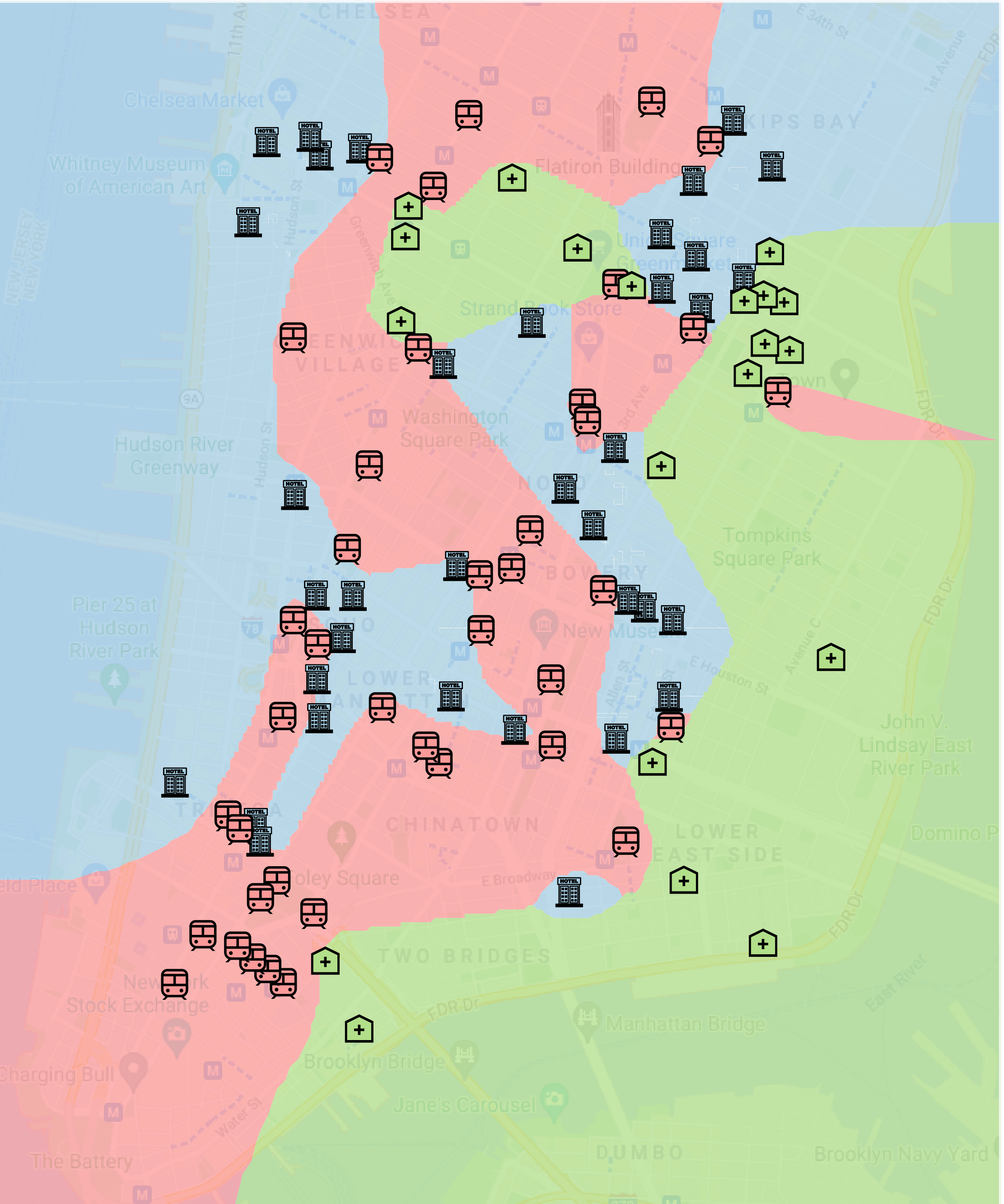


Delaunay Triangulation

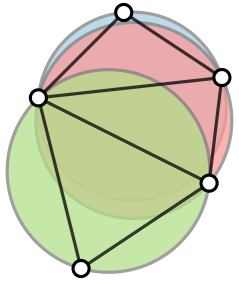


Gabriel Graph

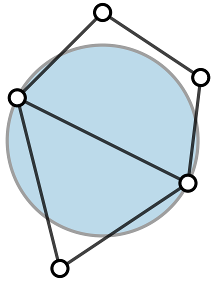
Pipeline



1. **Proximity Graph**

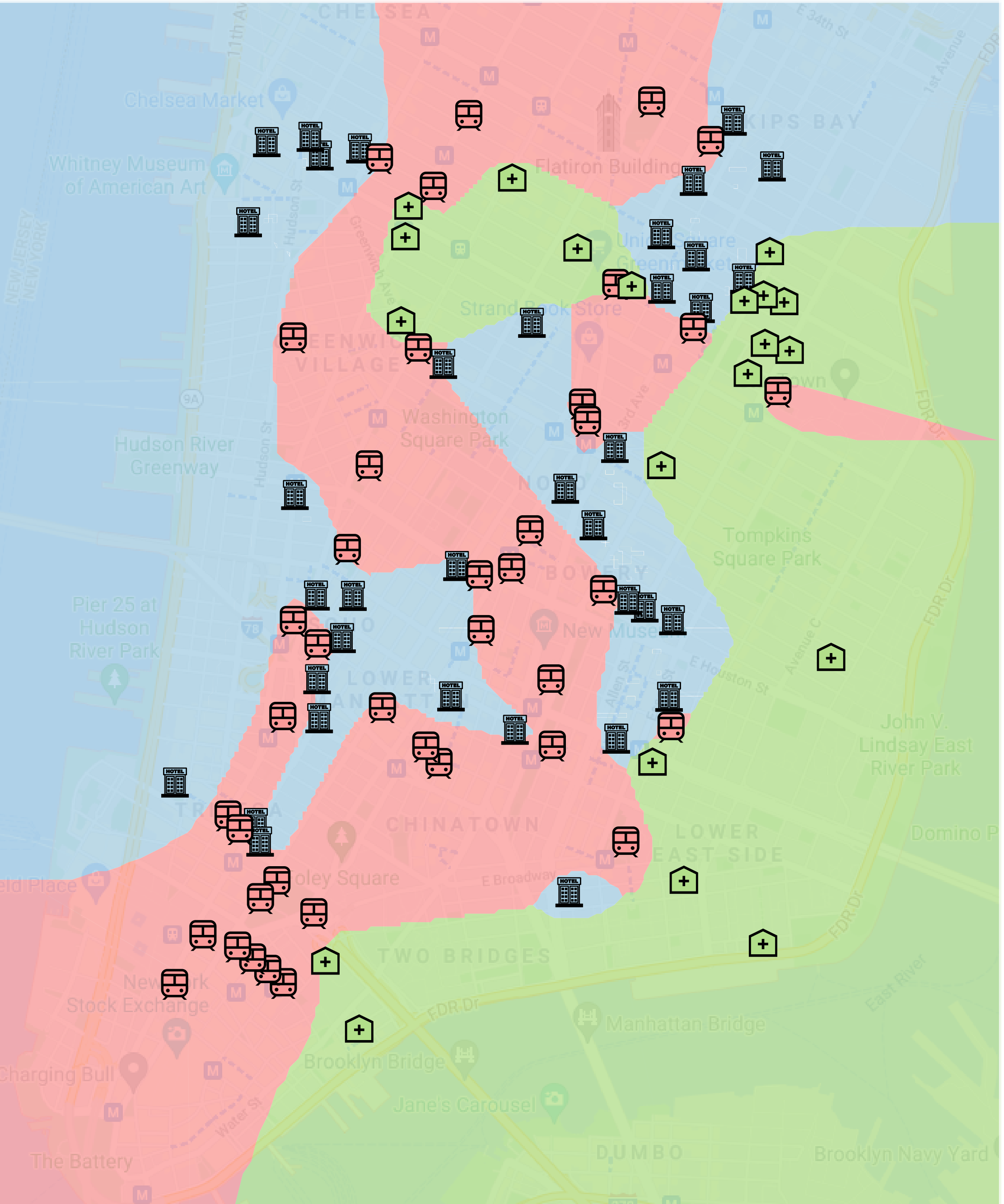


Delaunay Triangulation

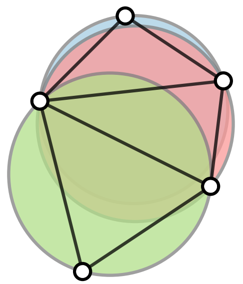


Gabriel Graph

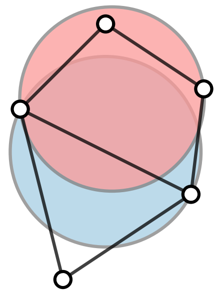
Pipeline



1. **Proximity Graph**

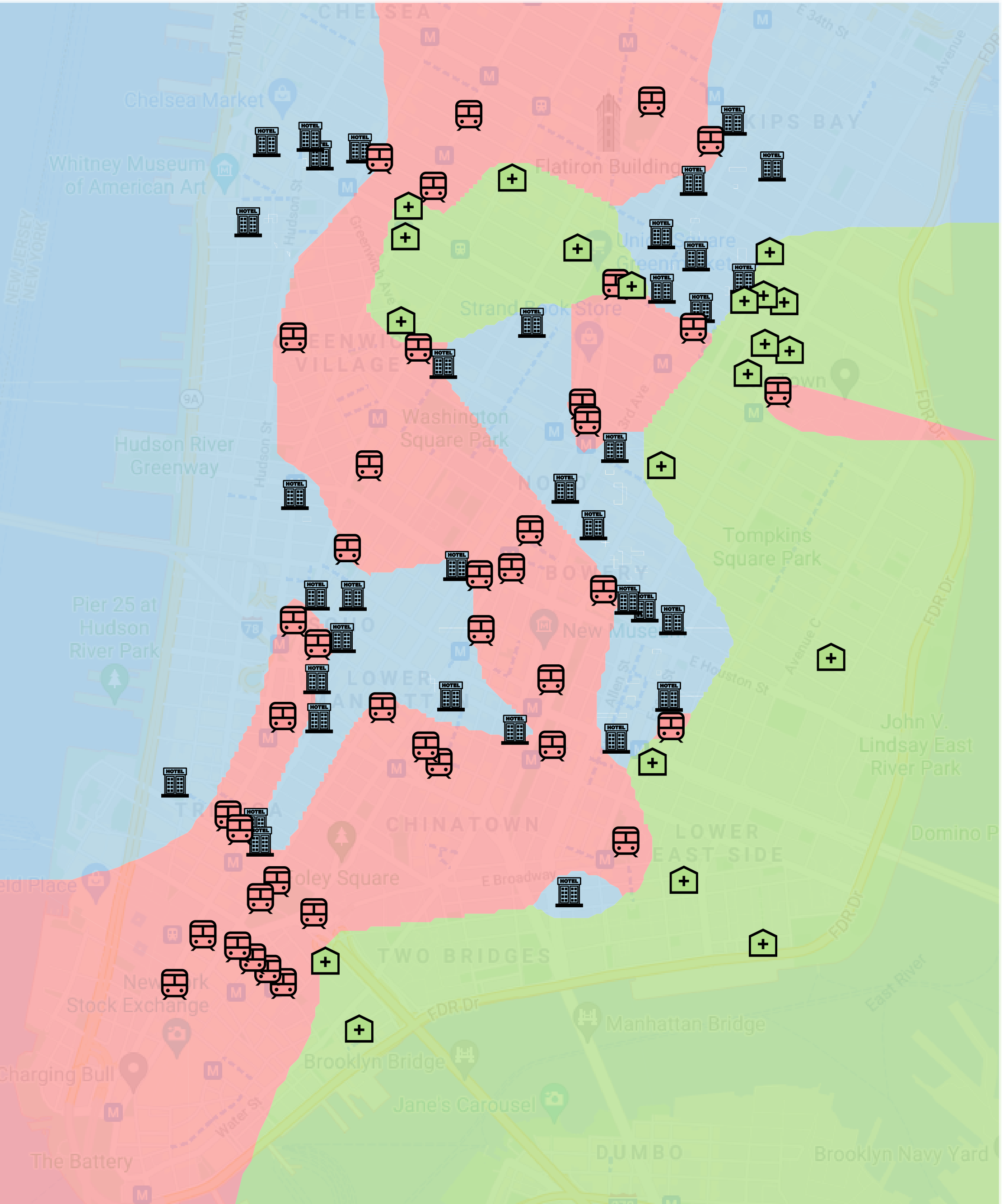


Delaunay Triangulation

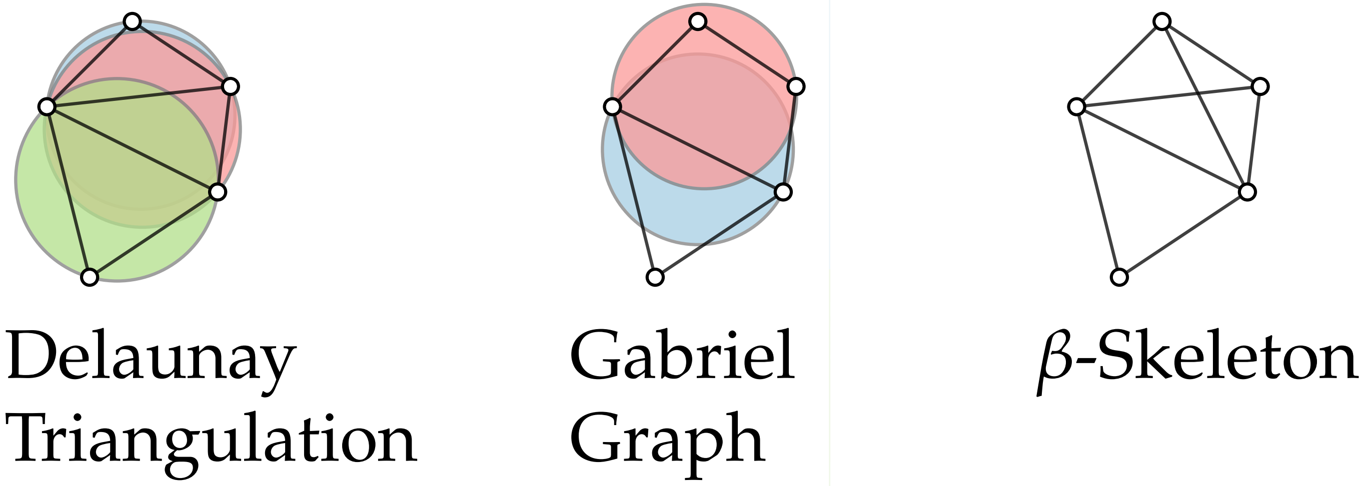


Gabriel Graph

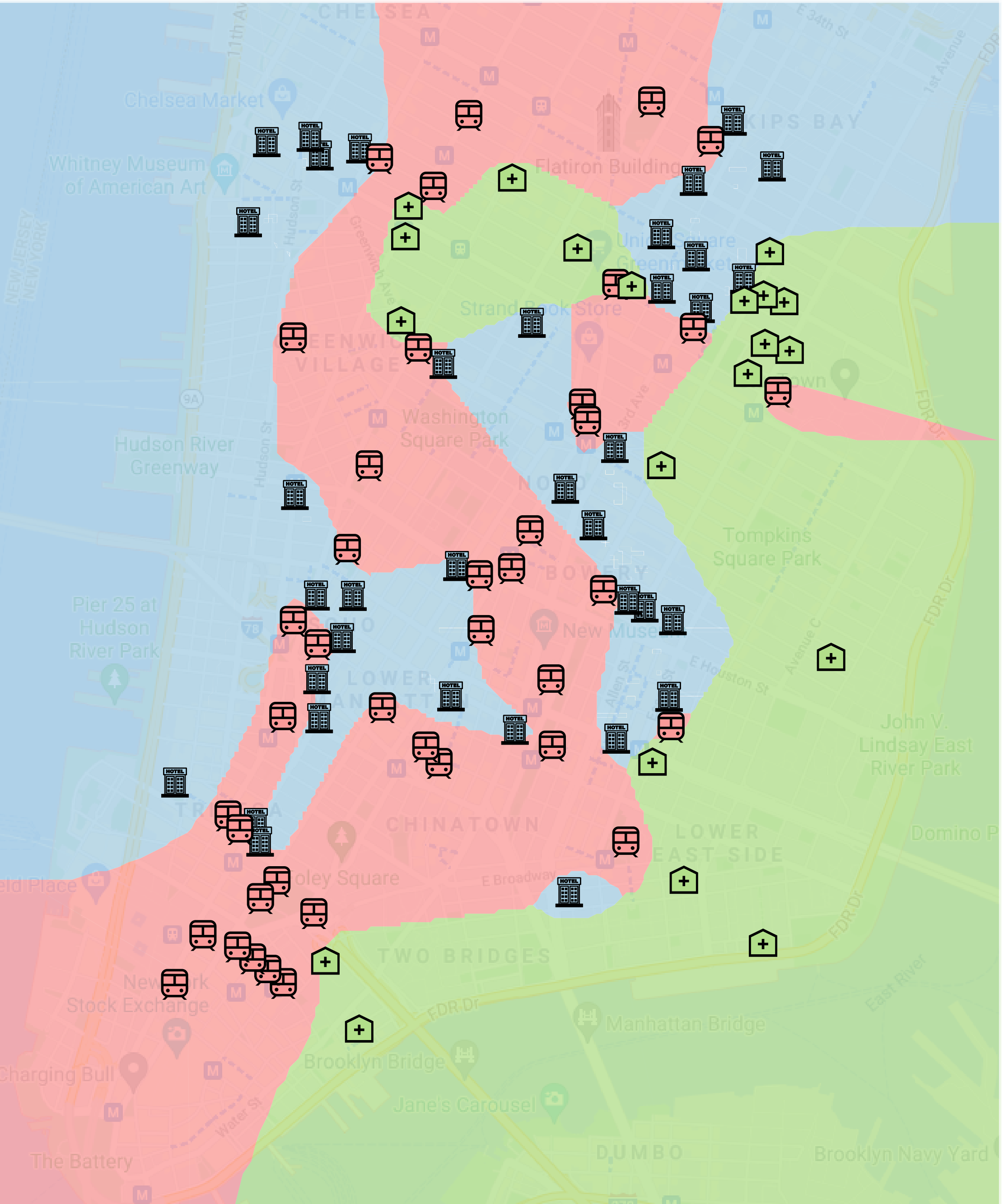
Pipeline



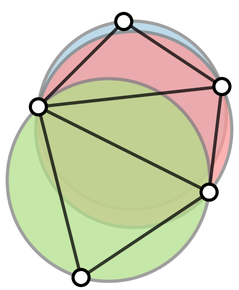
1. **Proximity Graph**



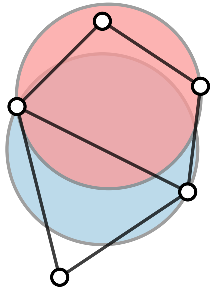
Pipeline



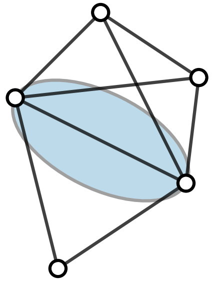
1. **Proximity Graph**



Delaunay Triangulation

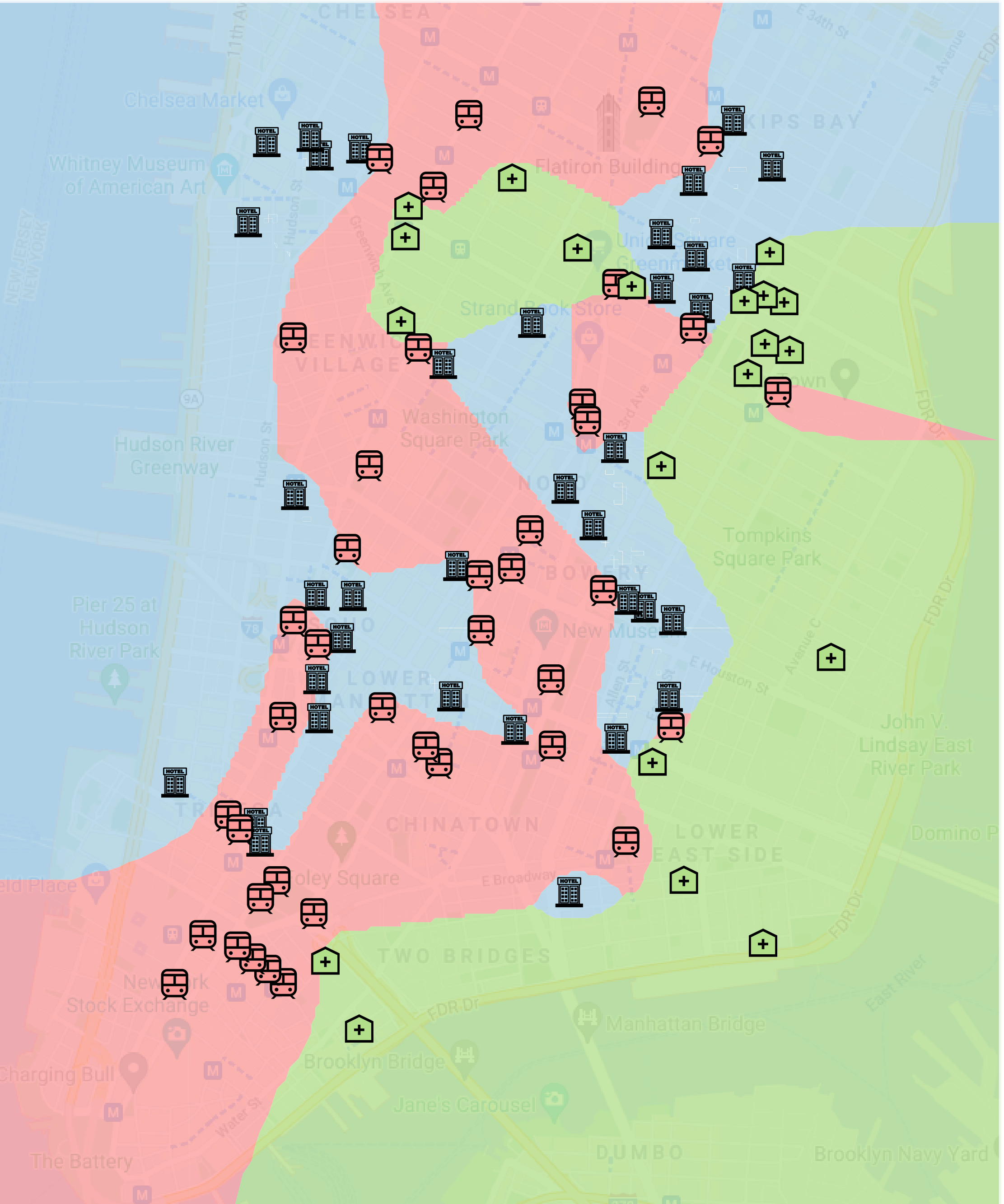


Gabriel Graph

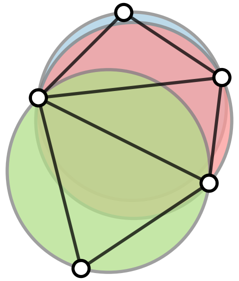


β-Skeleton

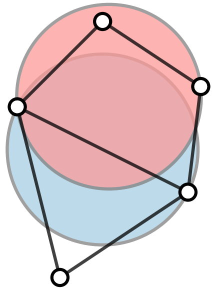
Pipeline



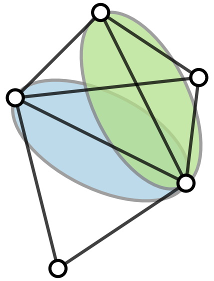
1. **Proximity Graph**



Delaunay Triangulation

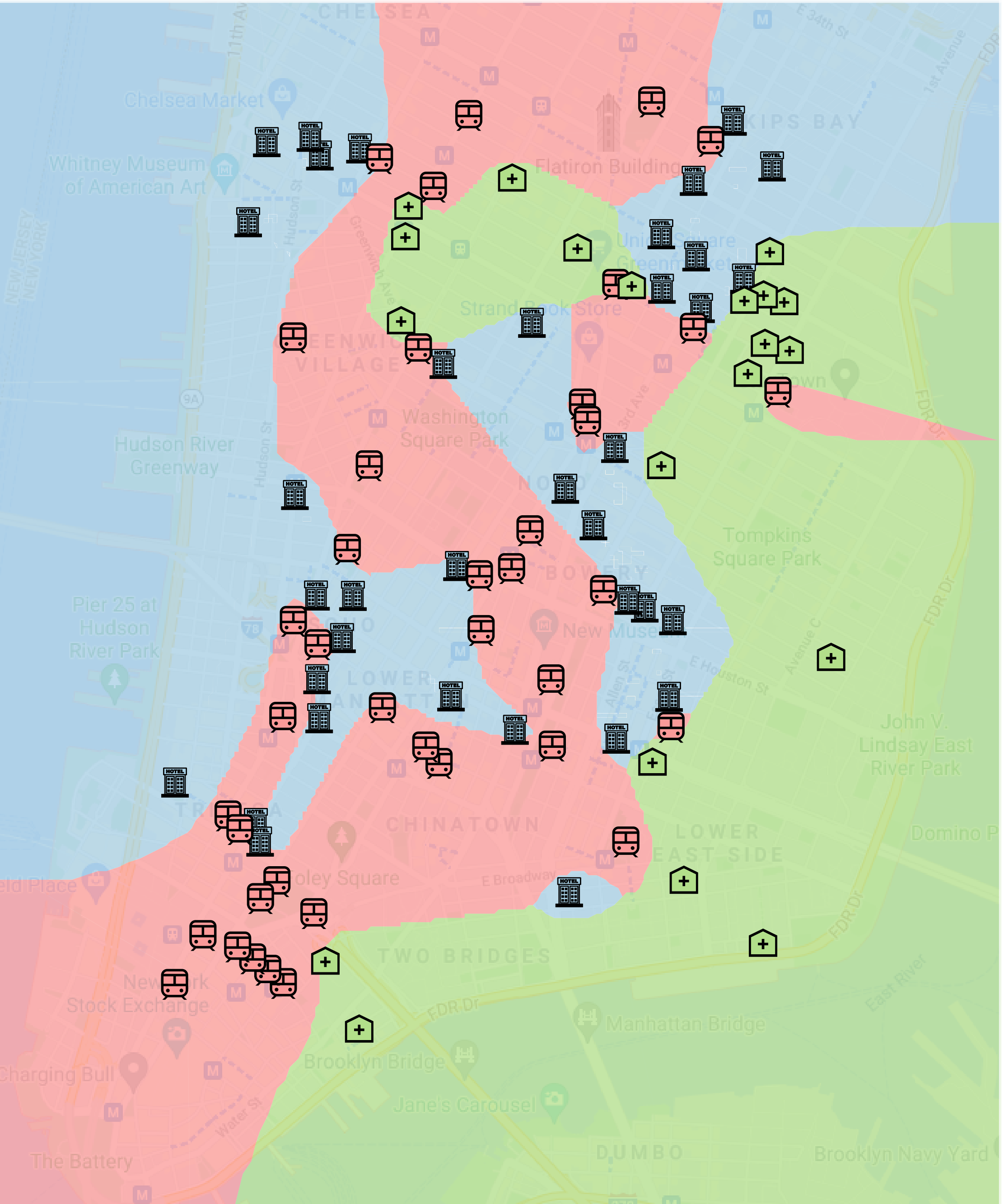


Gabriel Graph

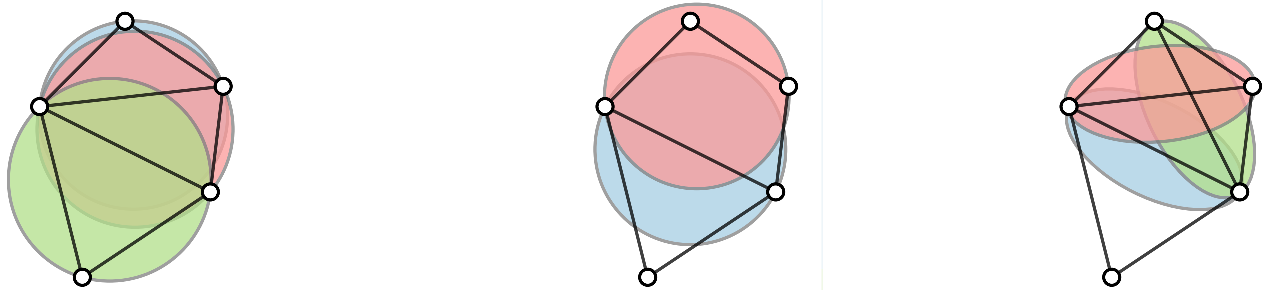


β-Skeleton

Pipeline



1. **Proximity Graph**

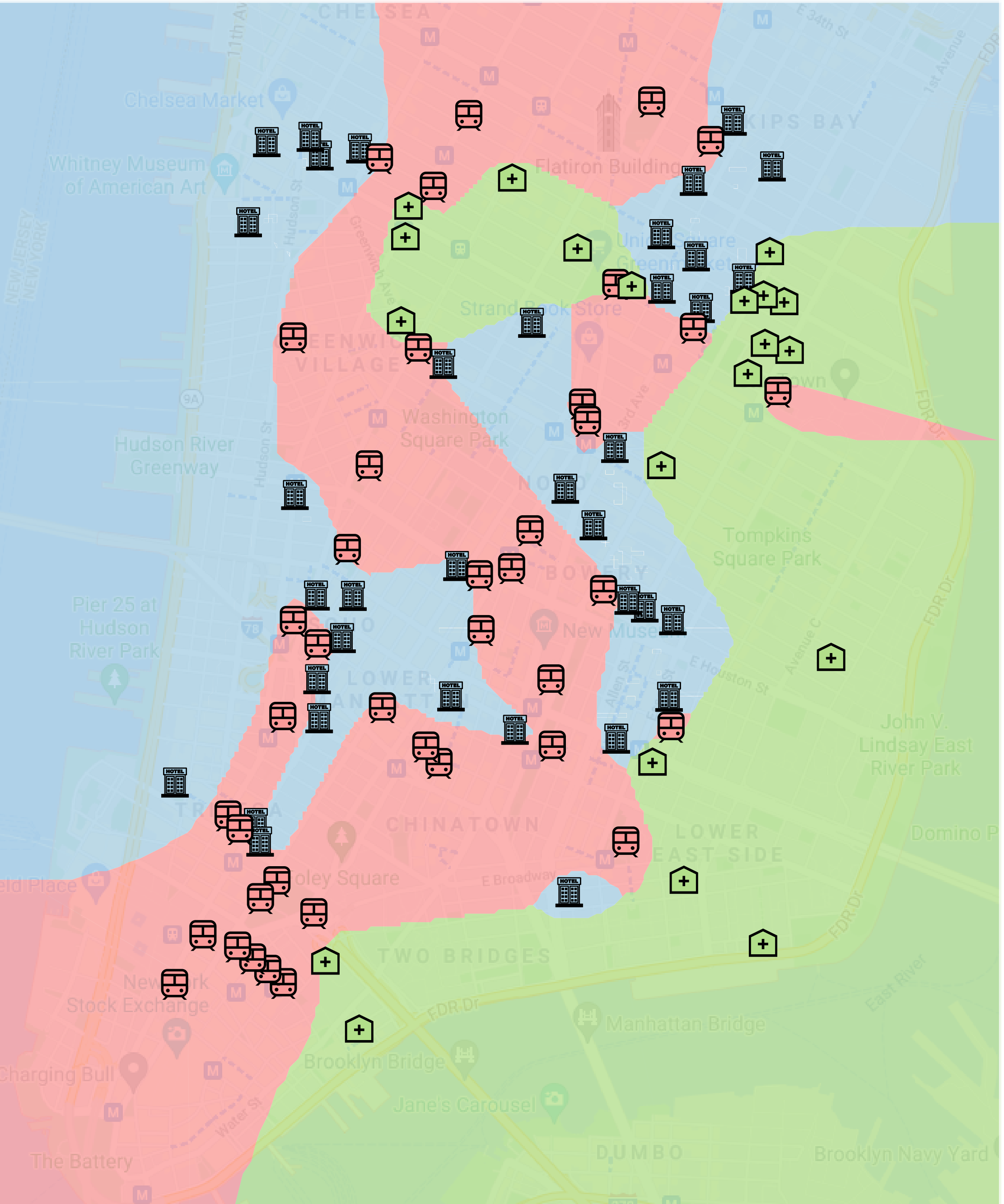


Delaunay Triangulation

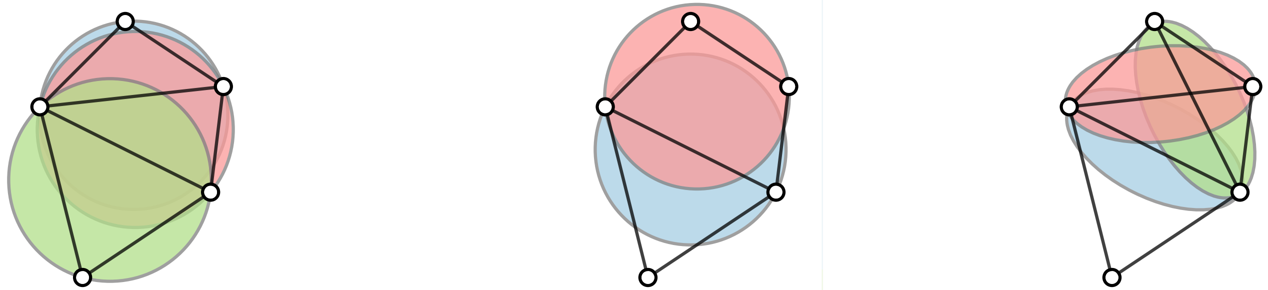
Gabriel Graph

β-Skeleton

Pipeline



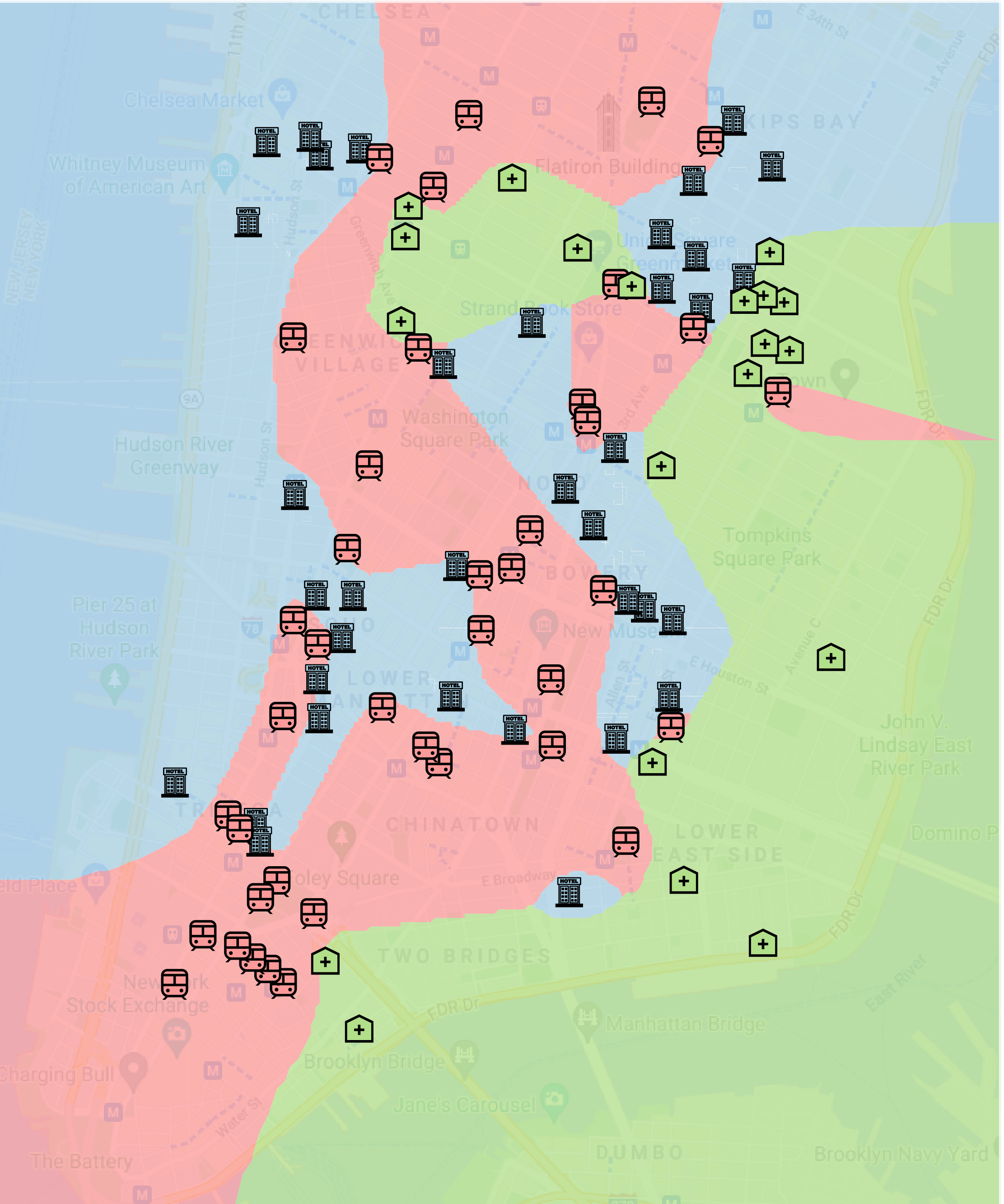
1. **Proximity Graph**



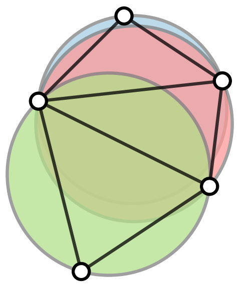
Delaunay Triangulation | Gabriel Graph | β-Skeleton

2. **Planar Spanning Forest**

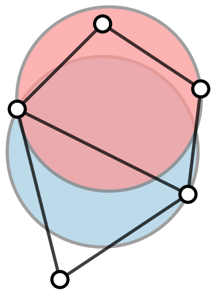
Pipeline



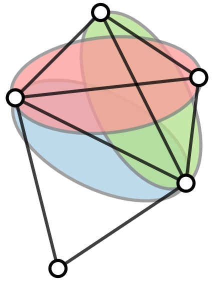
1. **Proximity Graph**



Delaunay Triangulation

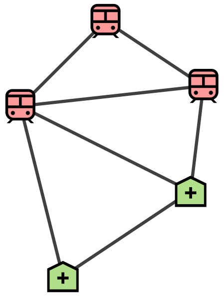


Gabriel Graph

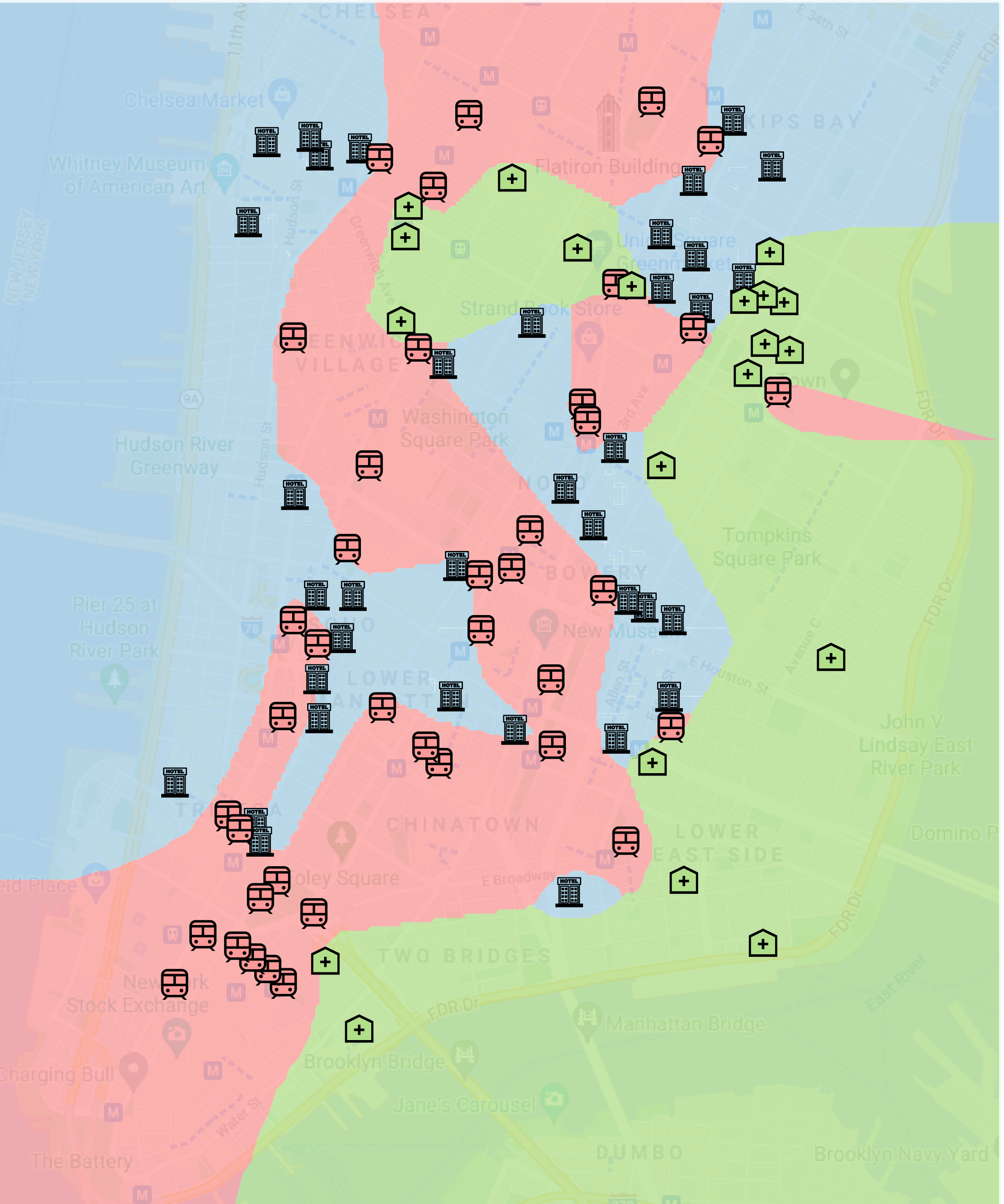


β-Skeleton

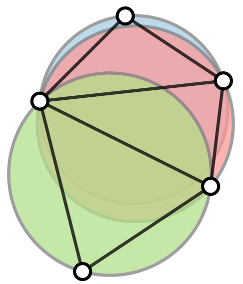
2. **Planar Spanning Forest**



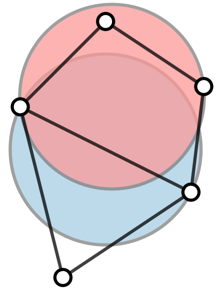
Pipeline



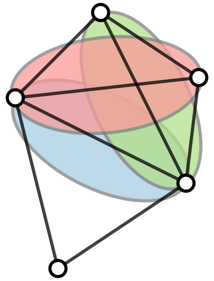
1. **Proximity Graph**



Delaunay Triangulation

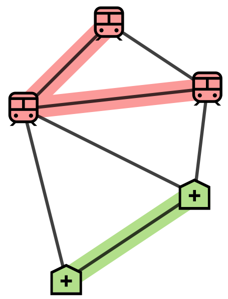


Gabriel Graph

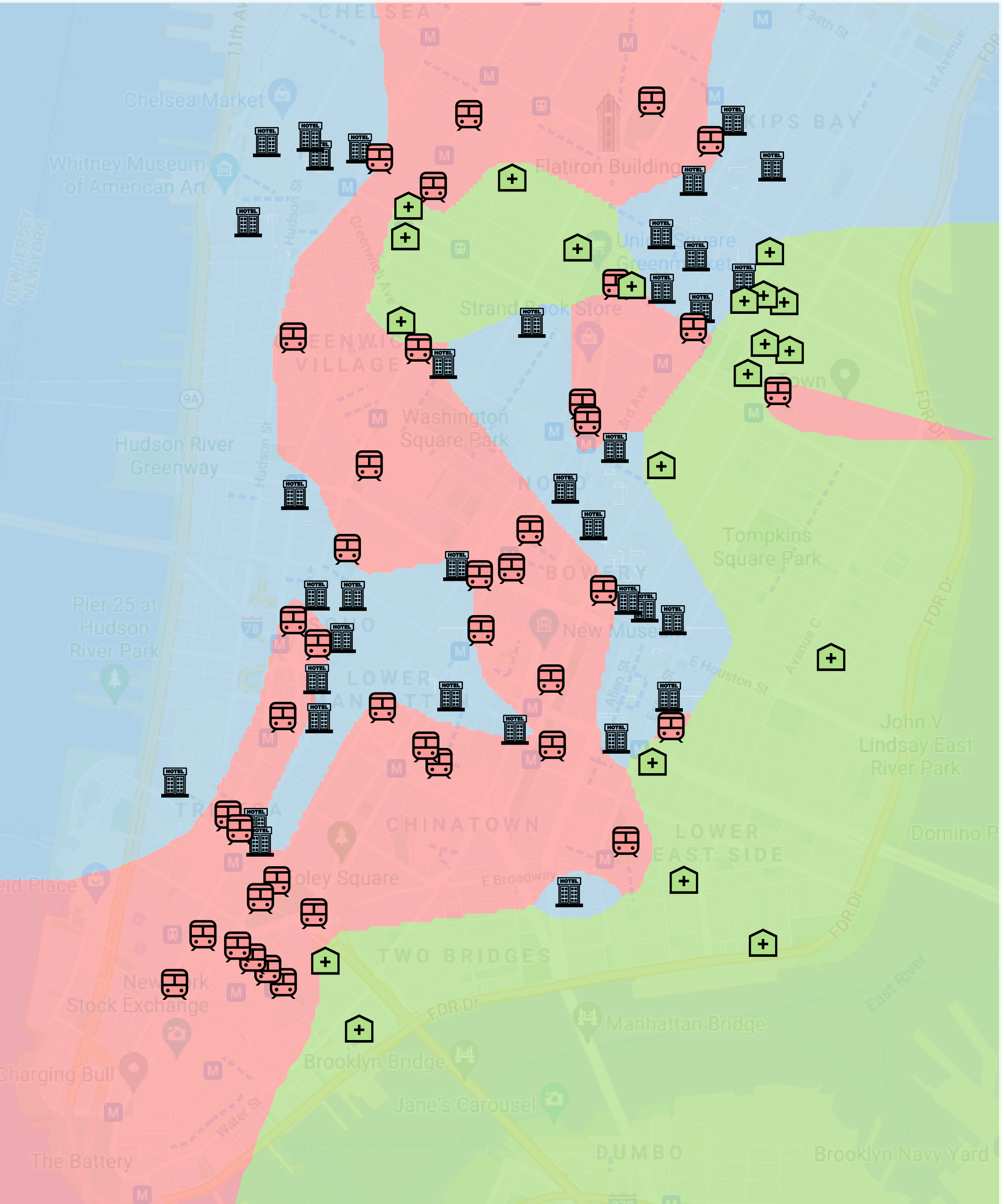


β-Skeleton

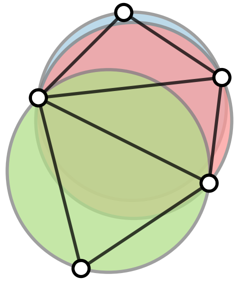
2. **Planar Spanning Forest**



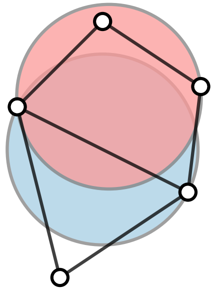
Pipeline



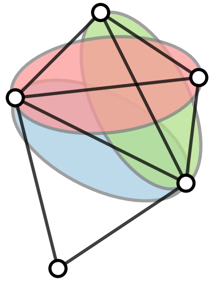
1. **Proximity Graph**



Delaunay Triangulation

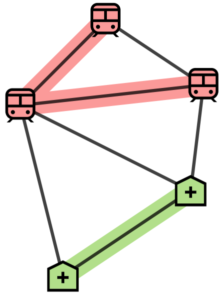


Gabriel Graph



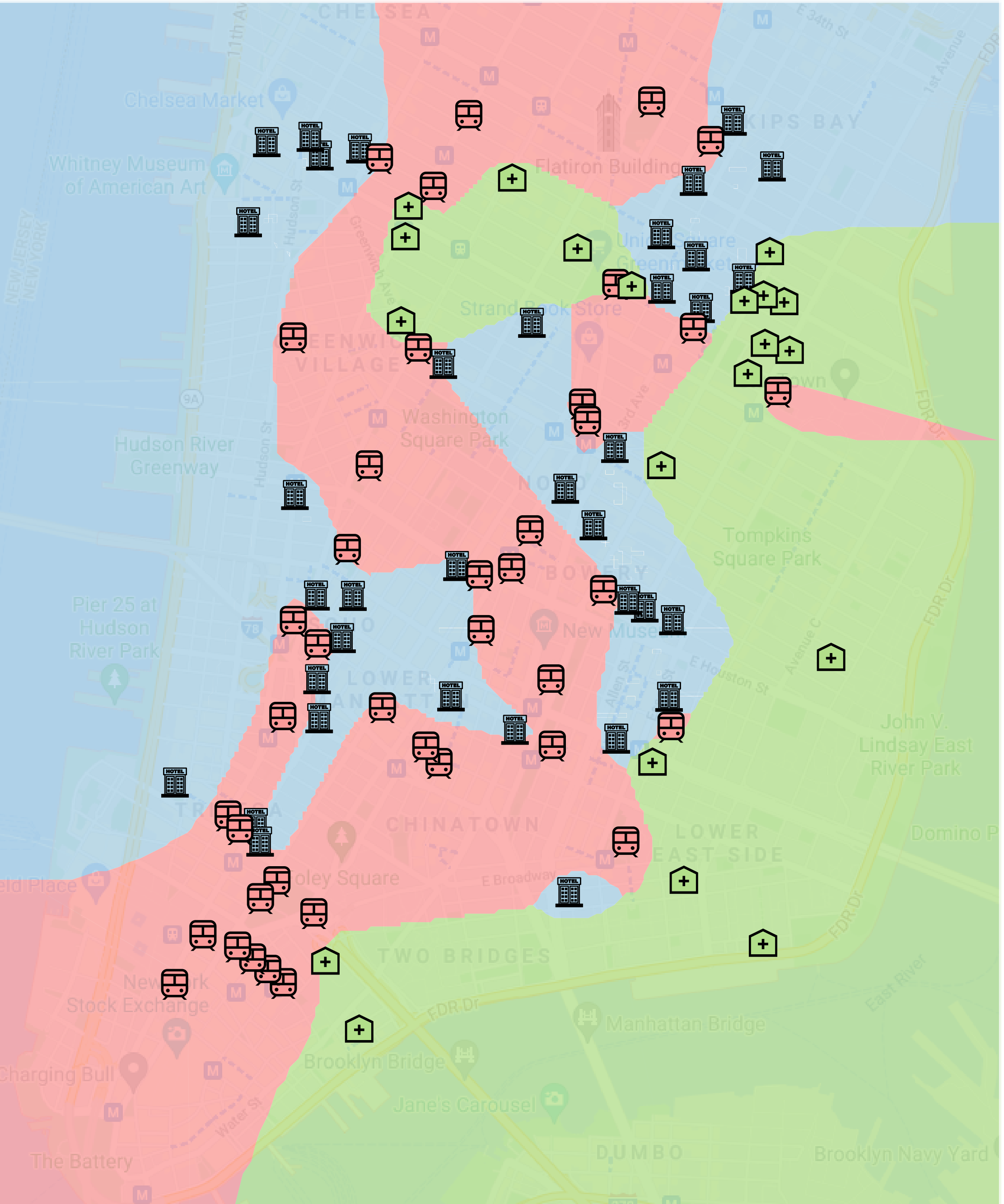
β-Skeleton

2. **Planar Spanning Forest**

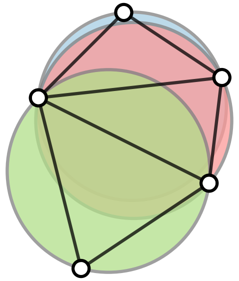


3. **Edge Augmentation**

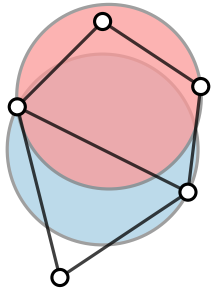
Pipeline



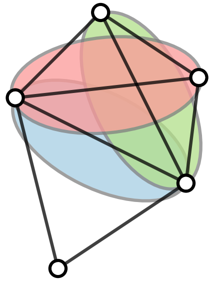
1. **Proximity Graph**



Delaunay Triangulation

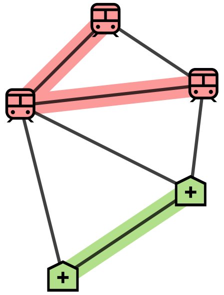


Gabriel Graph

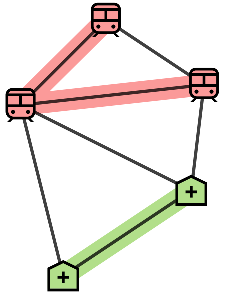


β-Skeleton

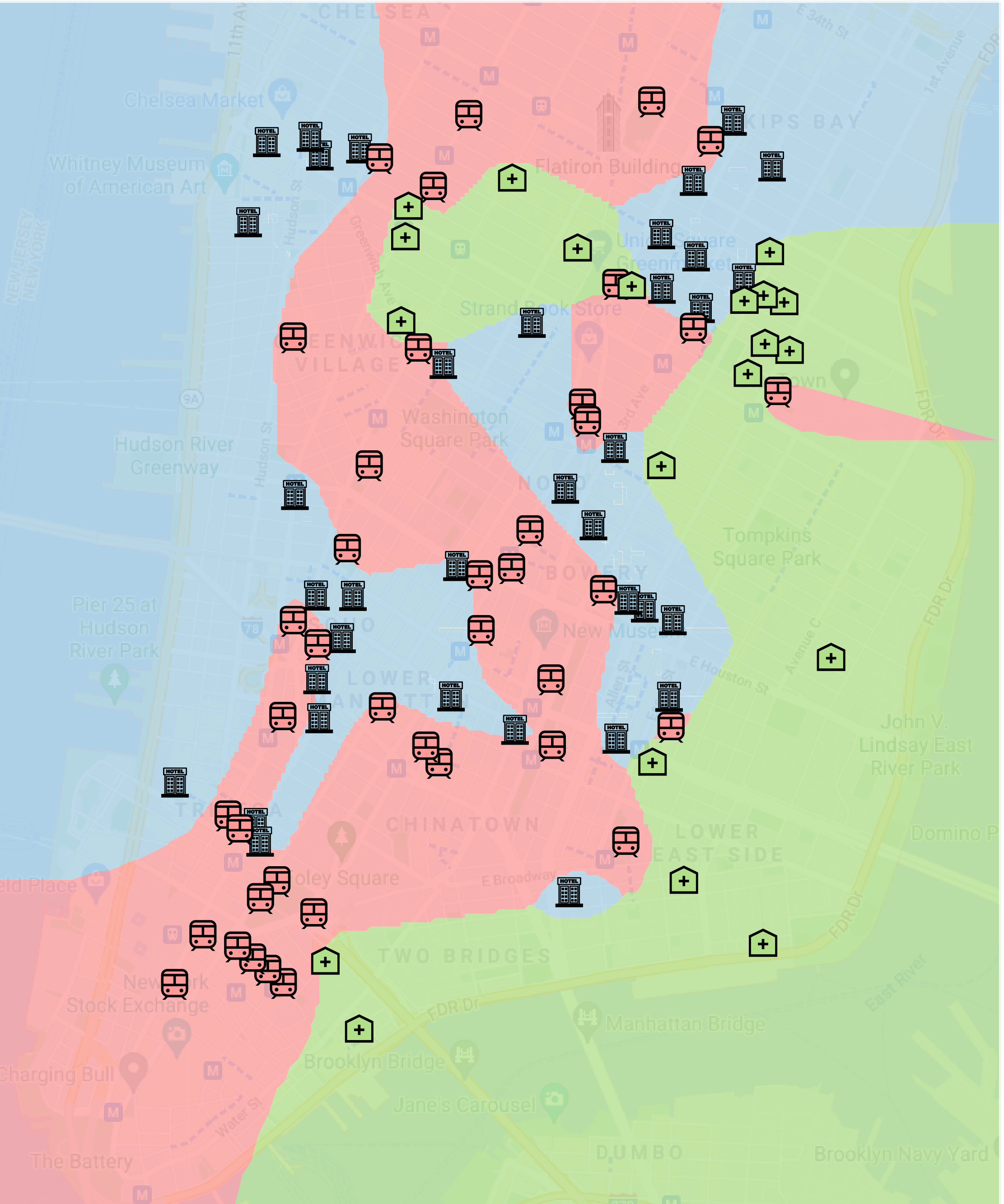
2. **Planar Spanning Forest**



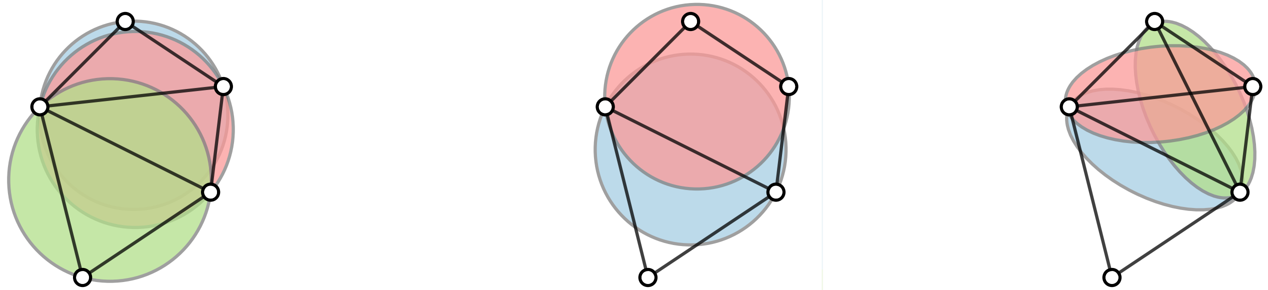
3. **Edge Augmentation**



Pipeline



1. **Proximity Graph**

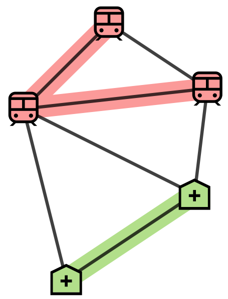


Delaunay Triangulation

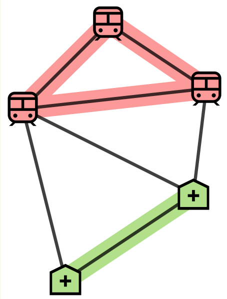
Gabriel Graph

β-Skeleton

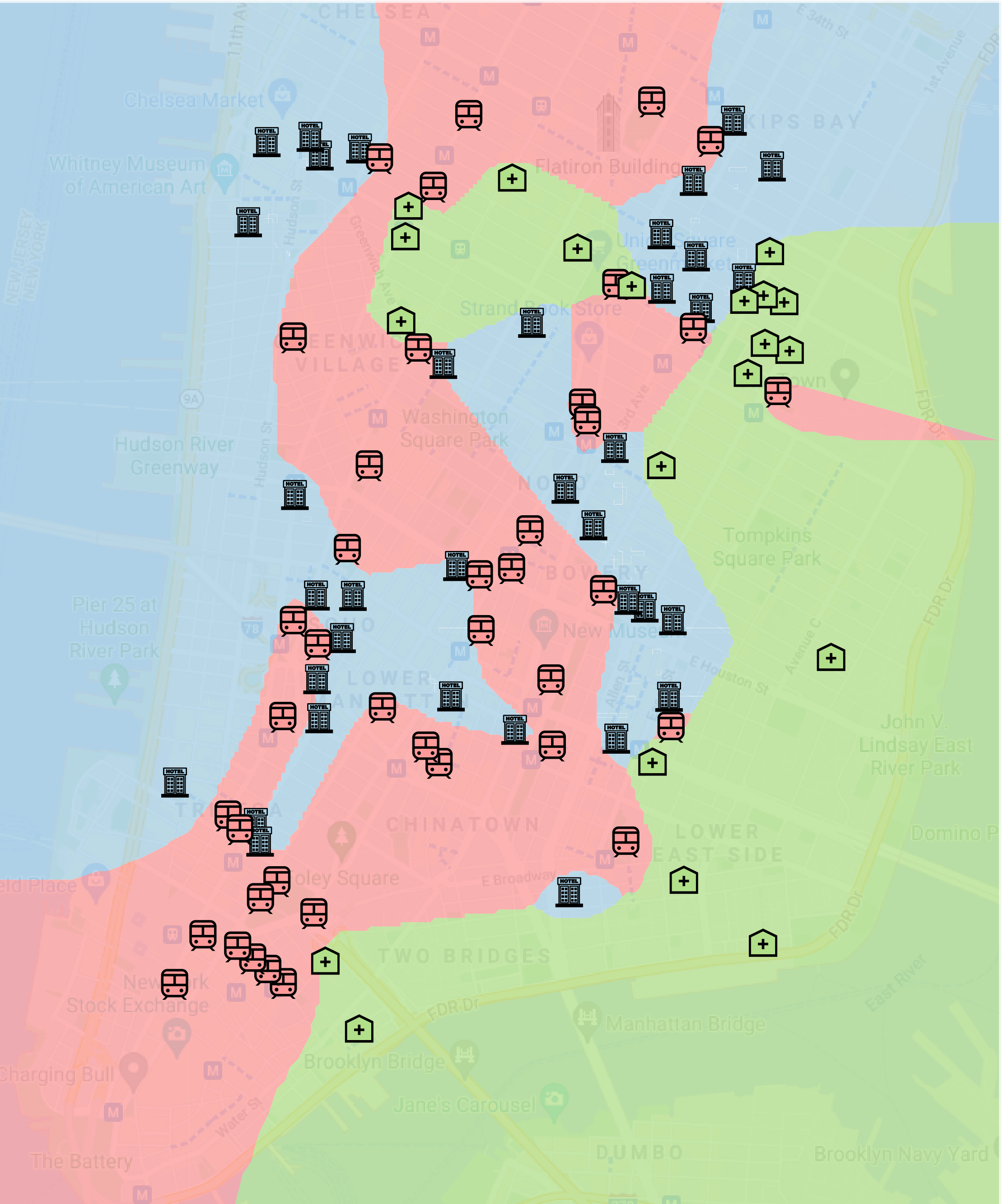
2. **Planar Spanning Forest**



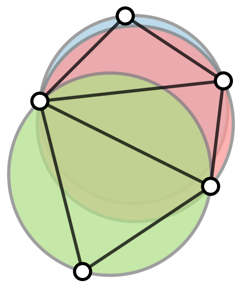
3. **Edge Augmentation**



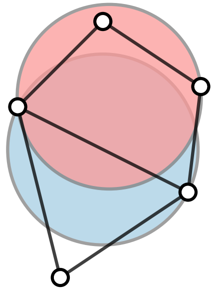
Pipeline



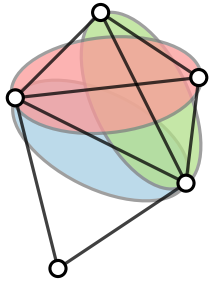
1. Proximity Graph



Delaunay Triangulation

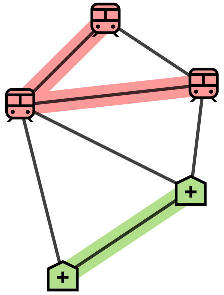


Gabriel Graph

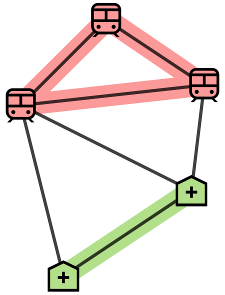


β-Skeleton

2. Planar Spanning Forest

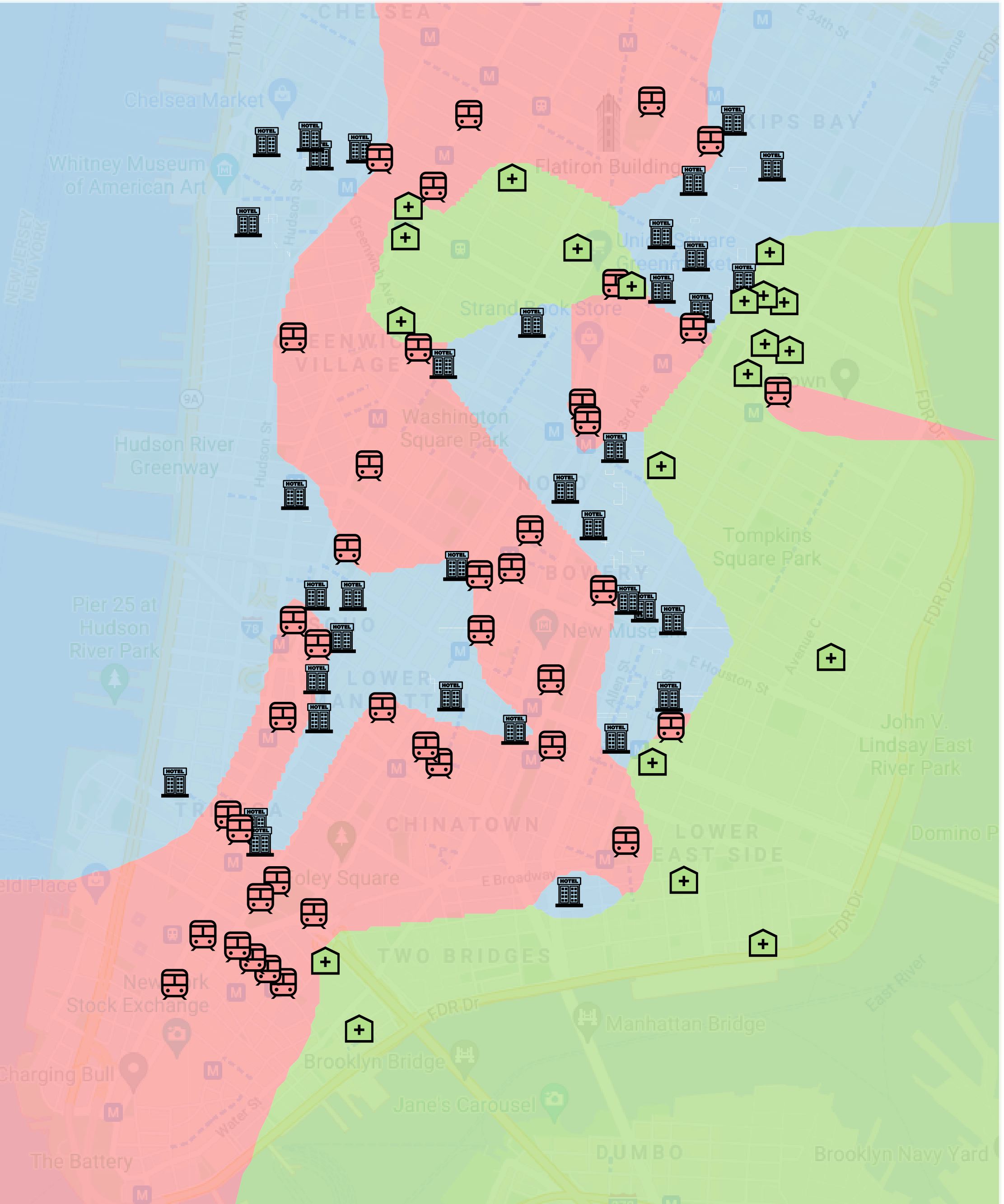


3. Edge Augmentation

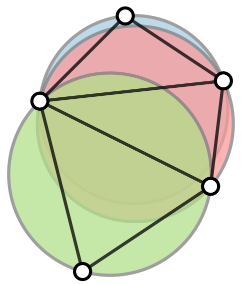


4. Rendering

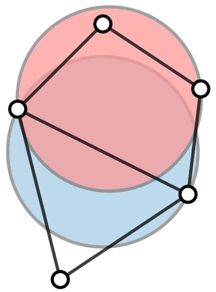
Pipeline



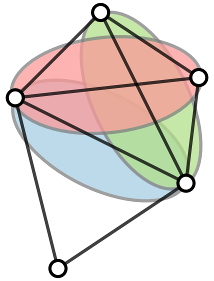
1. Proximity Graph



Delaunay Triangulation

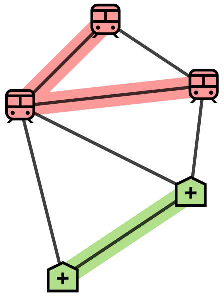


Gabriel Graph

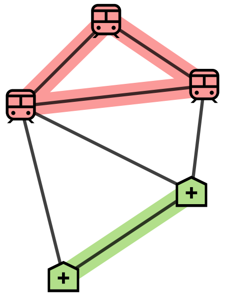


β-Skeleton

2. Planar Spanning Forest



3. Edge Augmentation

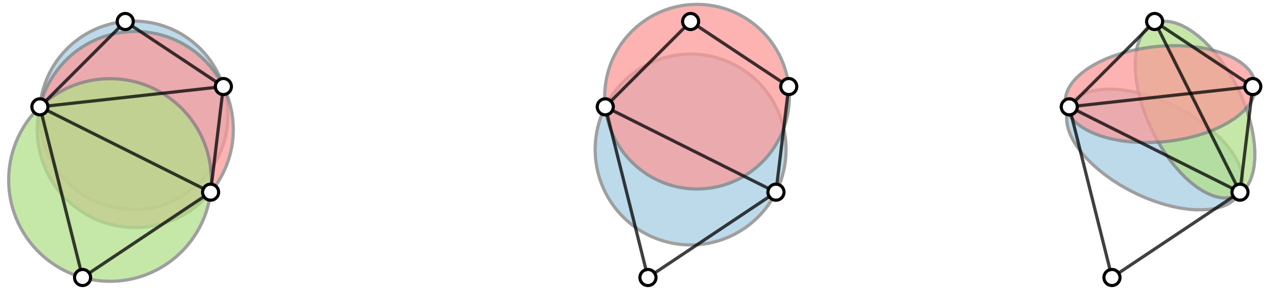


4. Rendering

- Line Voronoi Diagram

Pipeline

1. Proximity Graph

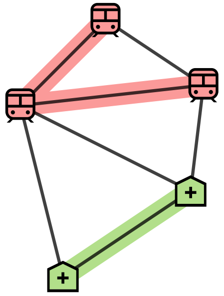


Delaunay Triangulation

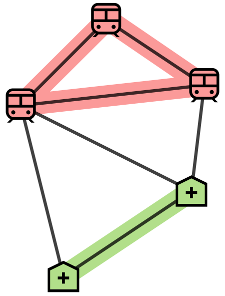
Gabriel Graph

β-Skeleton

2. Planar Spanning Forest



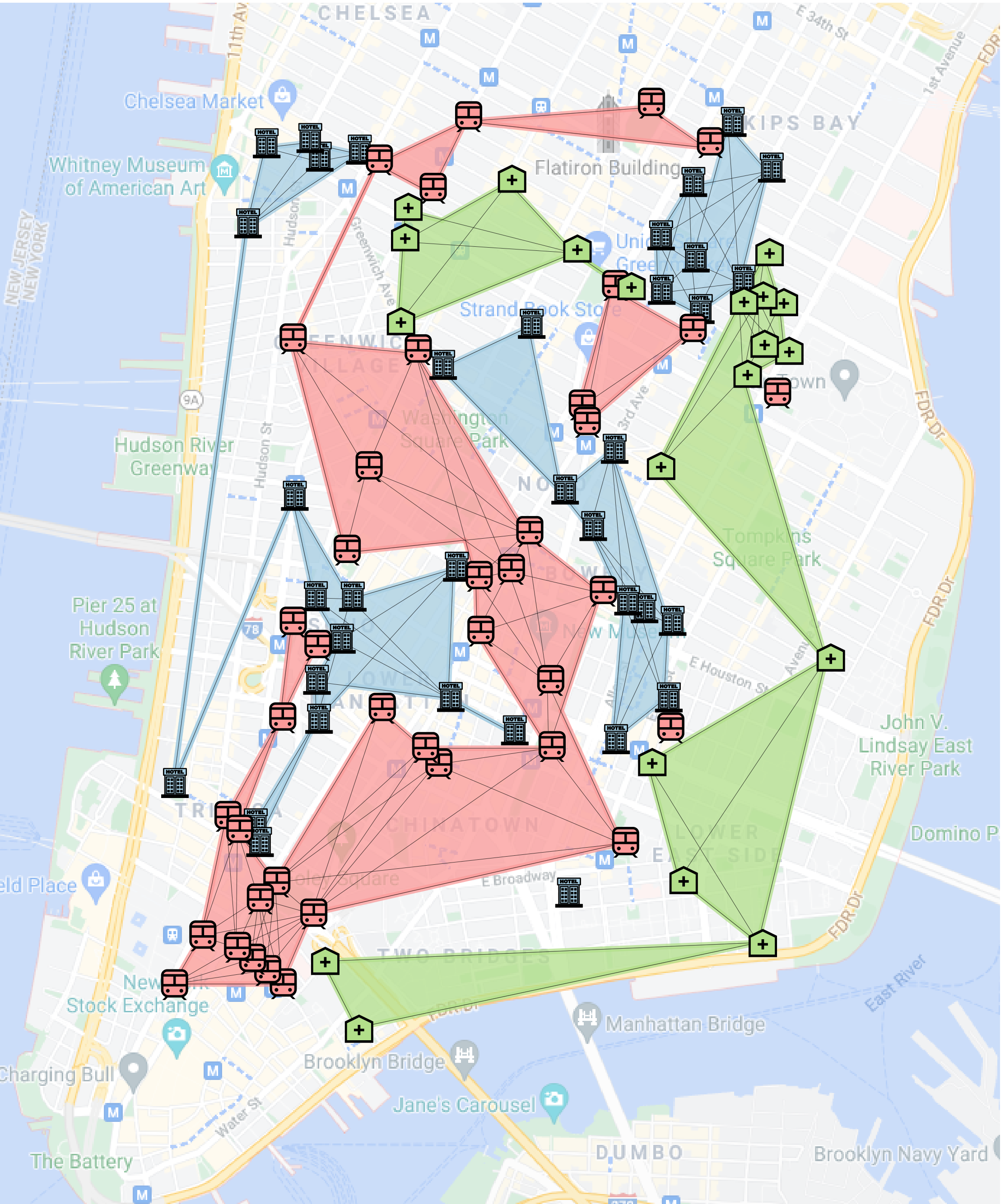
3. Edge Augmentation



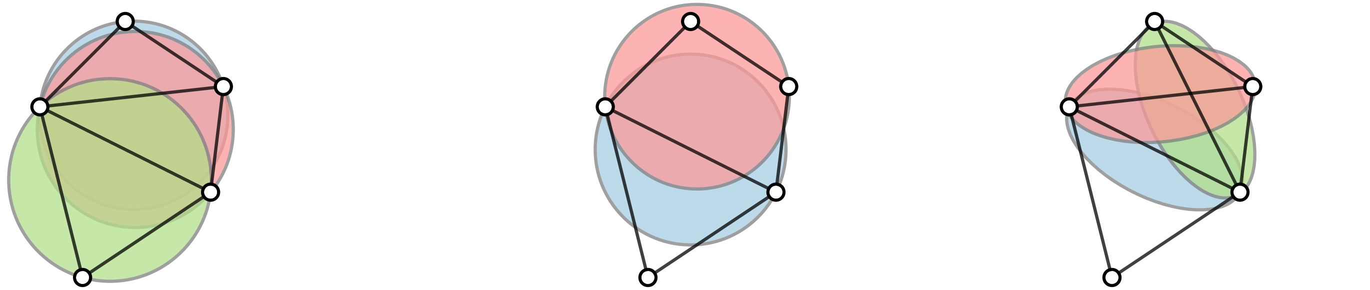
4. Rendering

- Line Voronoi Diagram
- Tree Representation

Pipeline



1. Proximity Graph

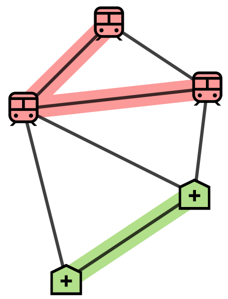


Delaunay Triangulation

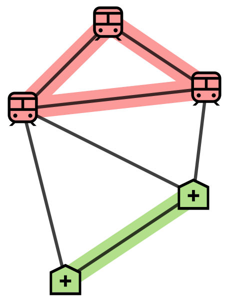
Gabriel Graph

β-Skeleton

2. Planar Spanning Forest



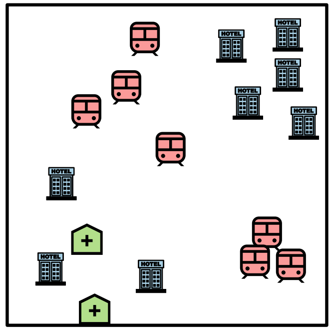
3. Edge Augmentation



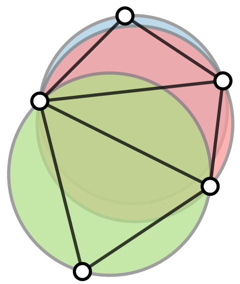
4. Rendering

- Line Voronoi Diagram
- Tree Representation
- Polygon Representation

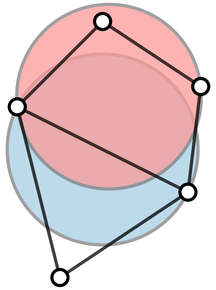
Pipeline



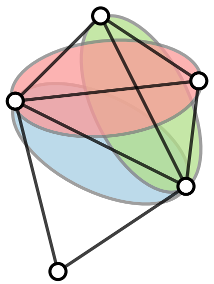
1. **Proximity Graph**



Delaunay Triangulation

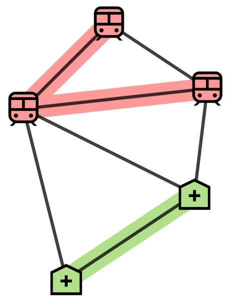


Gabriel Graph

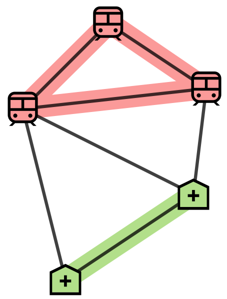


β-Skeleton

2. **Planar Spanning Forest**

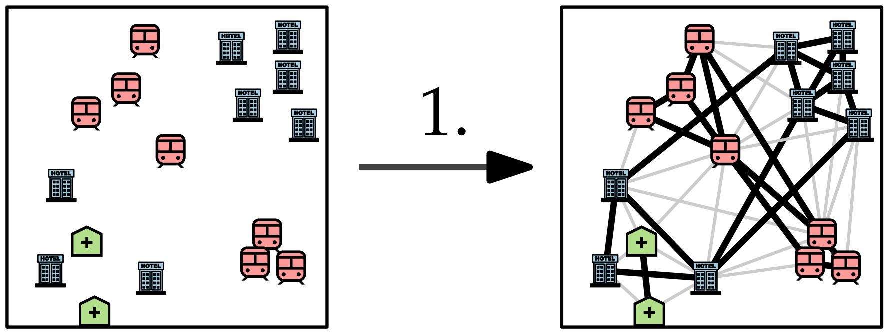


3. **Edge Augmentation**

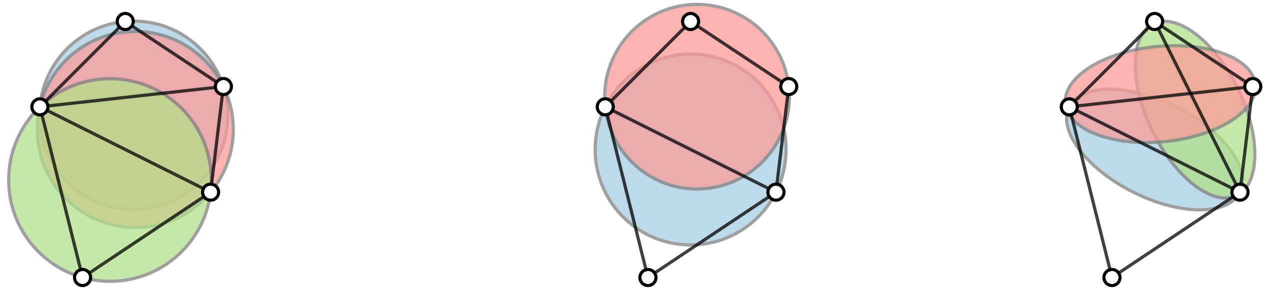


4. **Rendering**
   - Line Voronoi Diagram
   - Tree Representation
   - Polygon Representation

Pipeline

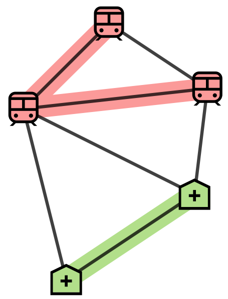


1. **Proximity Graph**

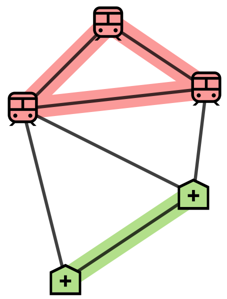


Delaunay Triangulation | Gabriel Graph | β-Skeleton

2. **Planar Spanning Forest**



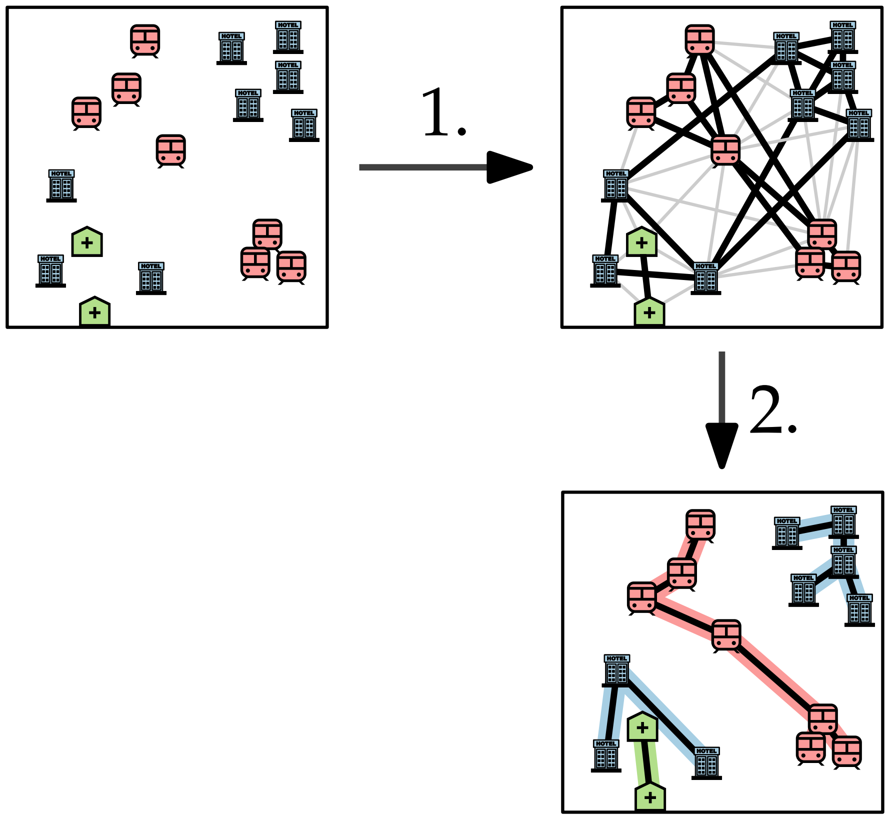
3. **Edge Augmentation**



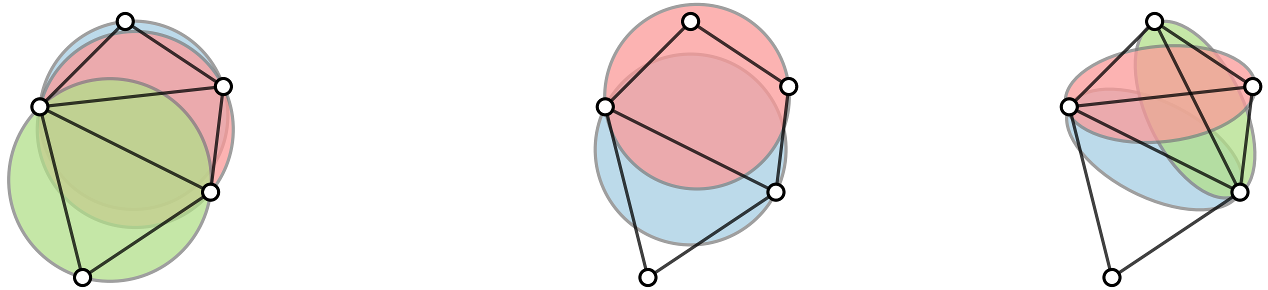
4. **Rendering**

- Line Voronoi Diagram
- Tree Representation
- Polygon Representation

Pipeline



1. **Proximity Graph**

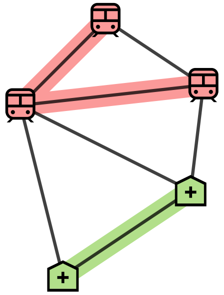


Delaunay Triangulation

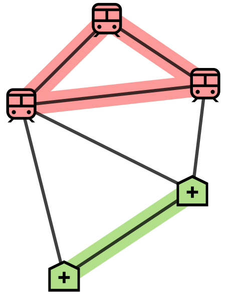
Gabriel Graph

β-Skeleton

2. **Planar Spanning Forest**



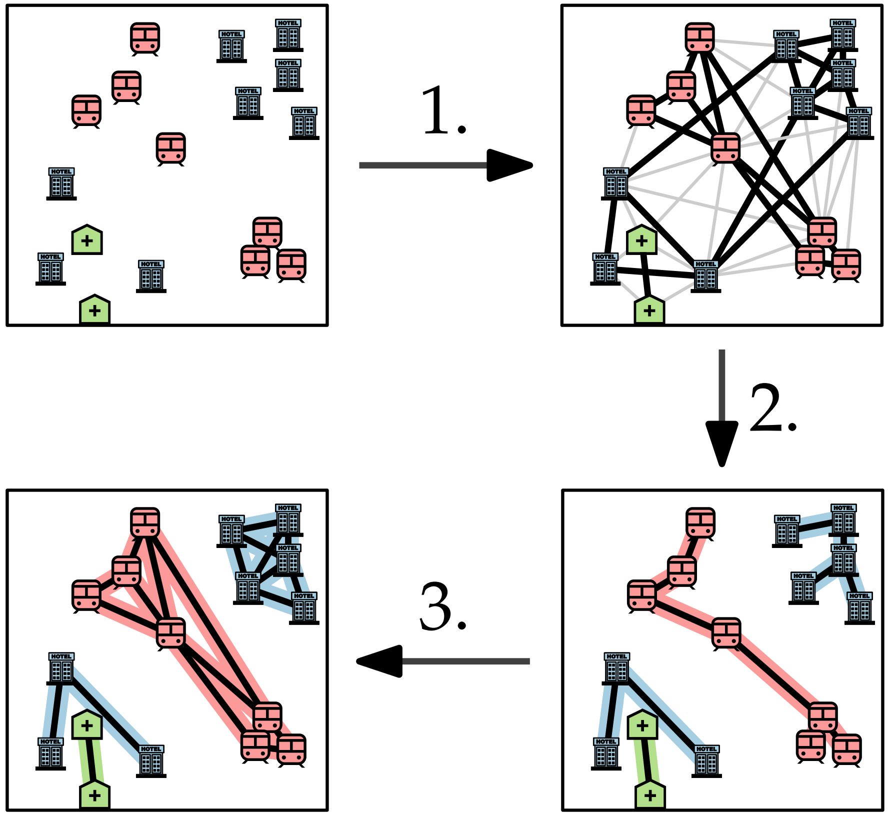
3. **Edge Augmentation**



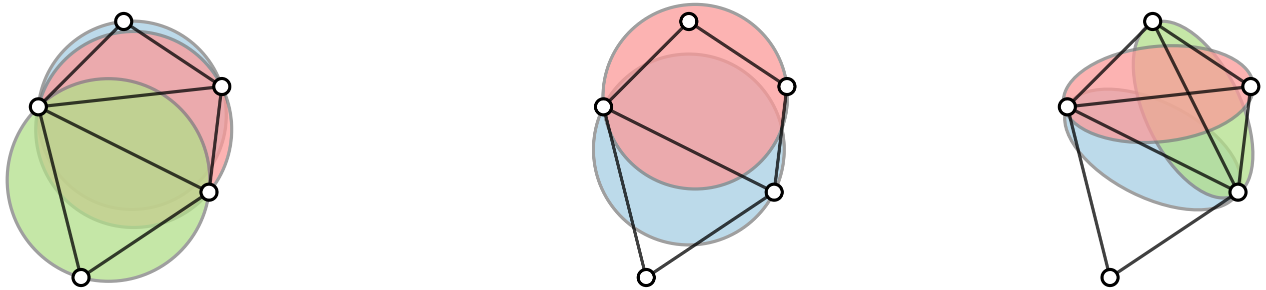
4. **Rendering**

- Line Voronoi Diagram
- Tree Representation
- Polygon Representation

Pipeline



1. Proximity Graph

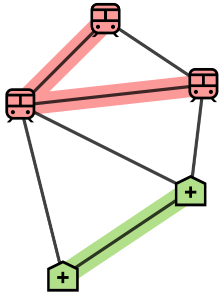


Delaunay Triangulation

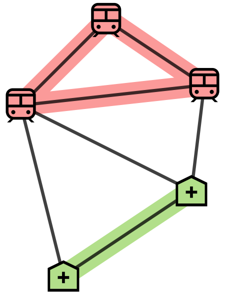
Gabriel Graph

β-Skeleton

2. Planar Spanning Forest

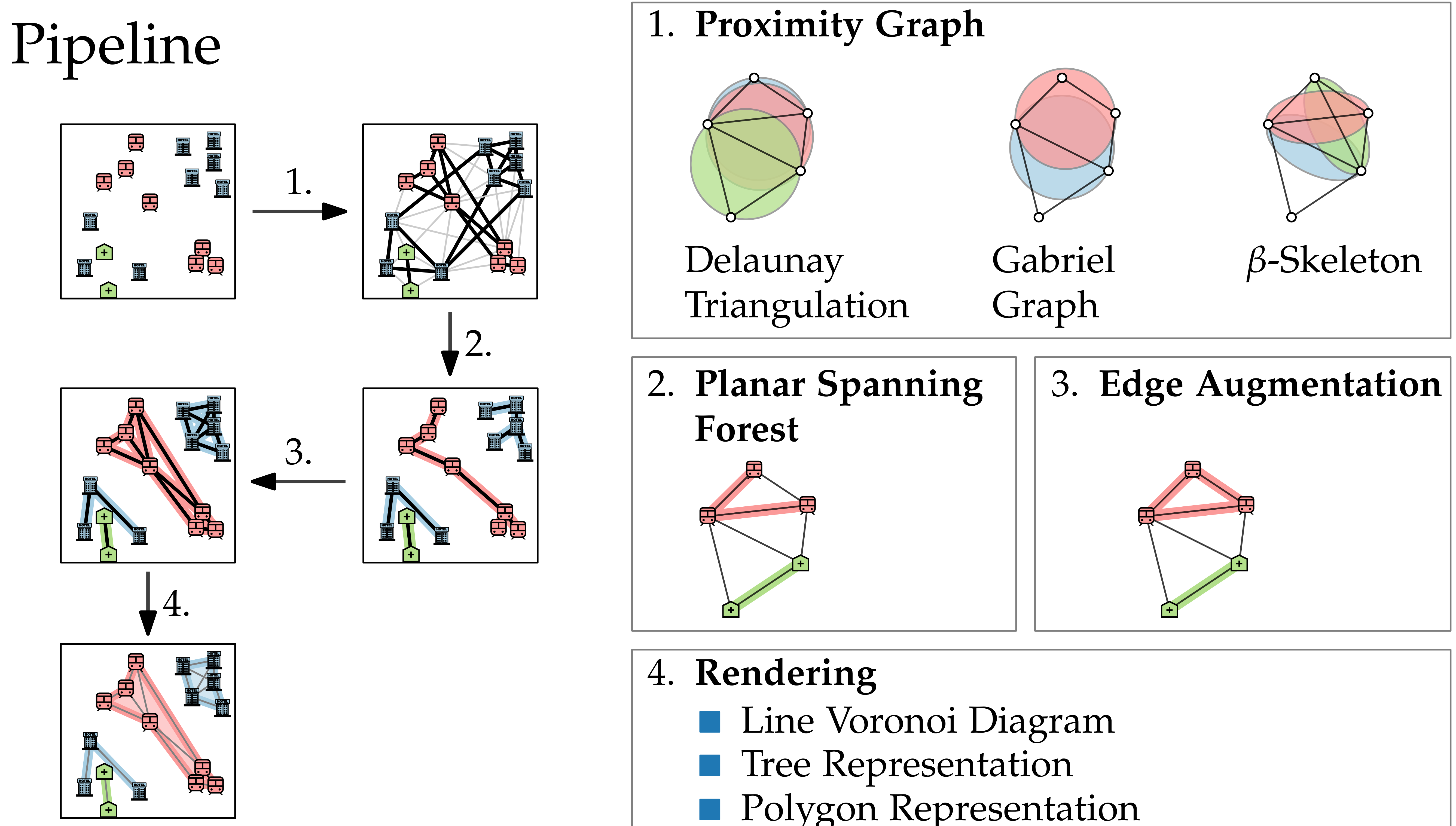


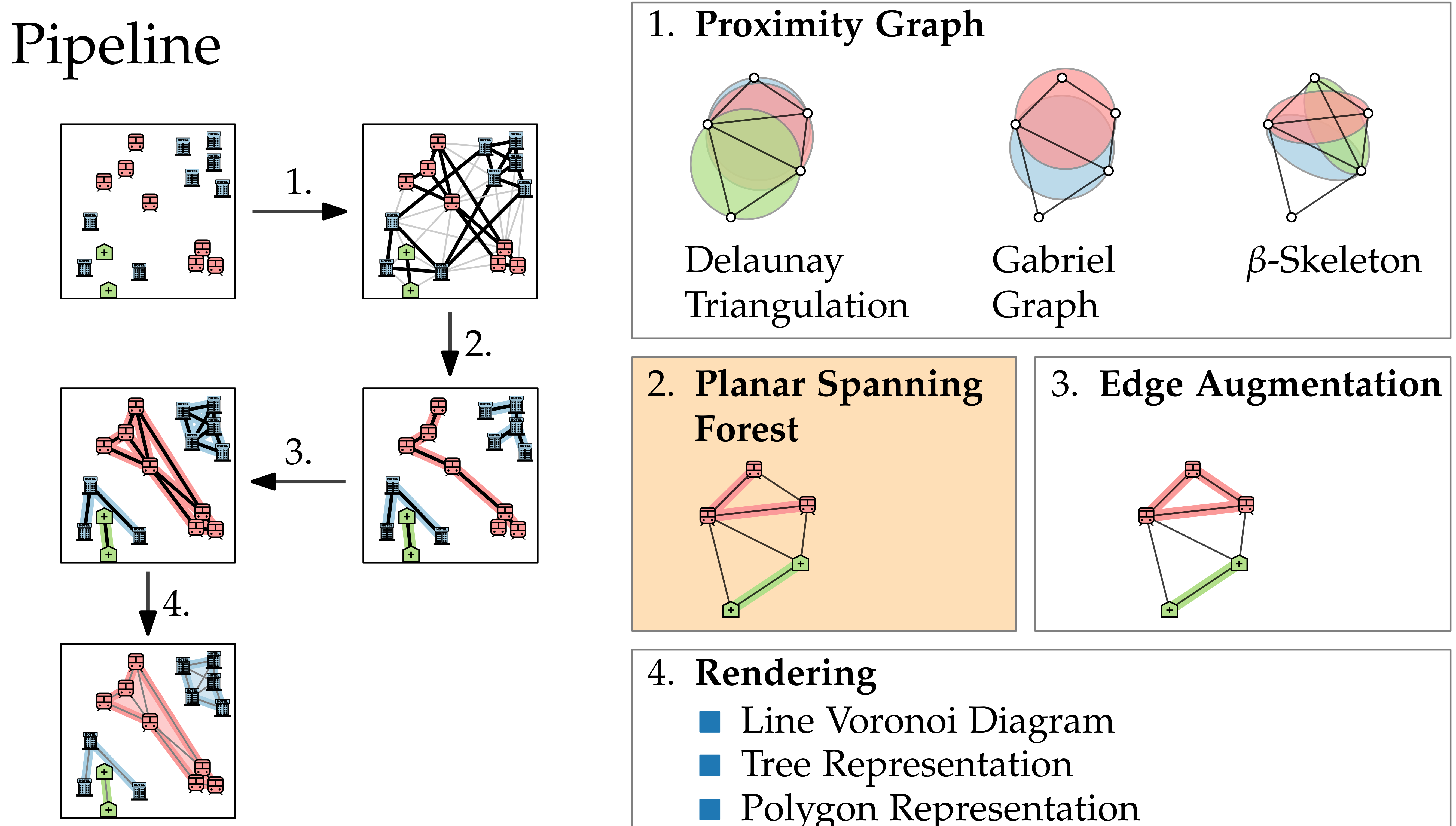
3. Edge Augmentation



4. Rendering

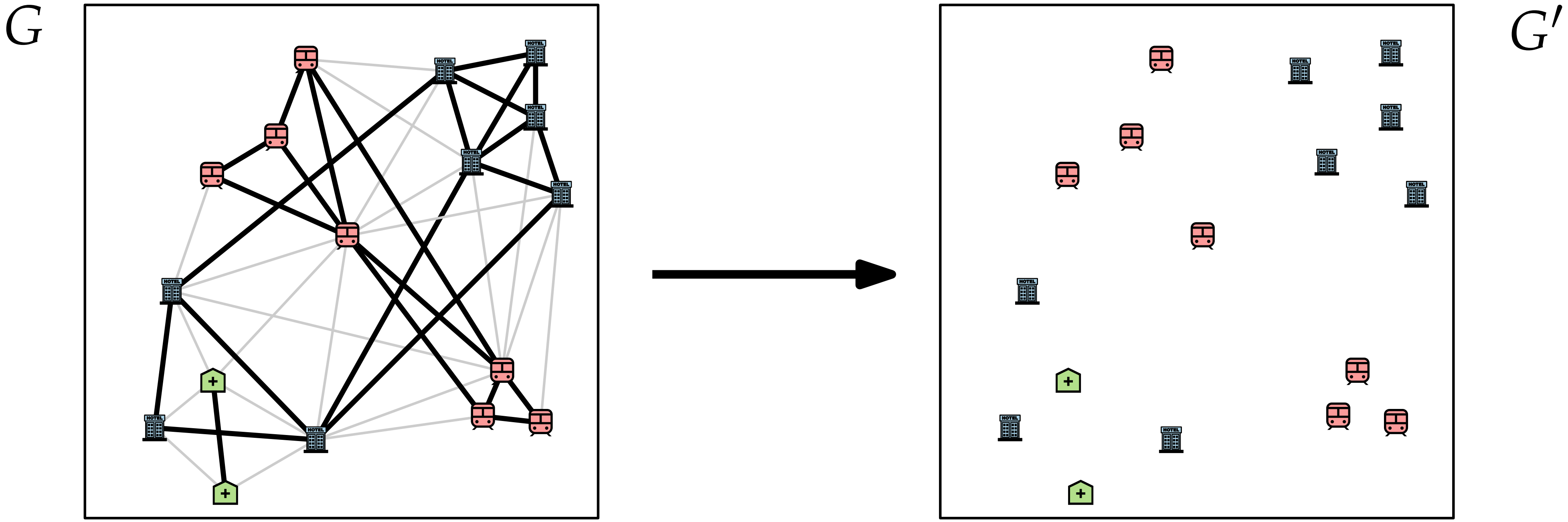
- Line Voronoi Diagram
- Tree Representation
- Polygon Representation


Pipeline
1.
2.
3.
4.
1. Proximity Graph
Delaunay Triangulation
Gabriel Graph
β-Skeleton
2. Planar Spanning Forest
3. Edge Augmentation
4. Rendering
Line Voronoi Diagram
Tree Representation
Polygon Representation

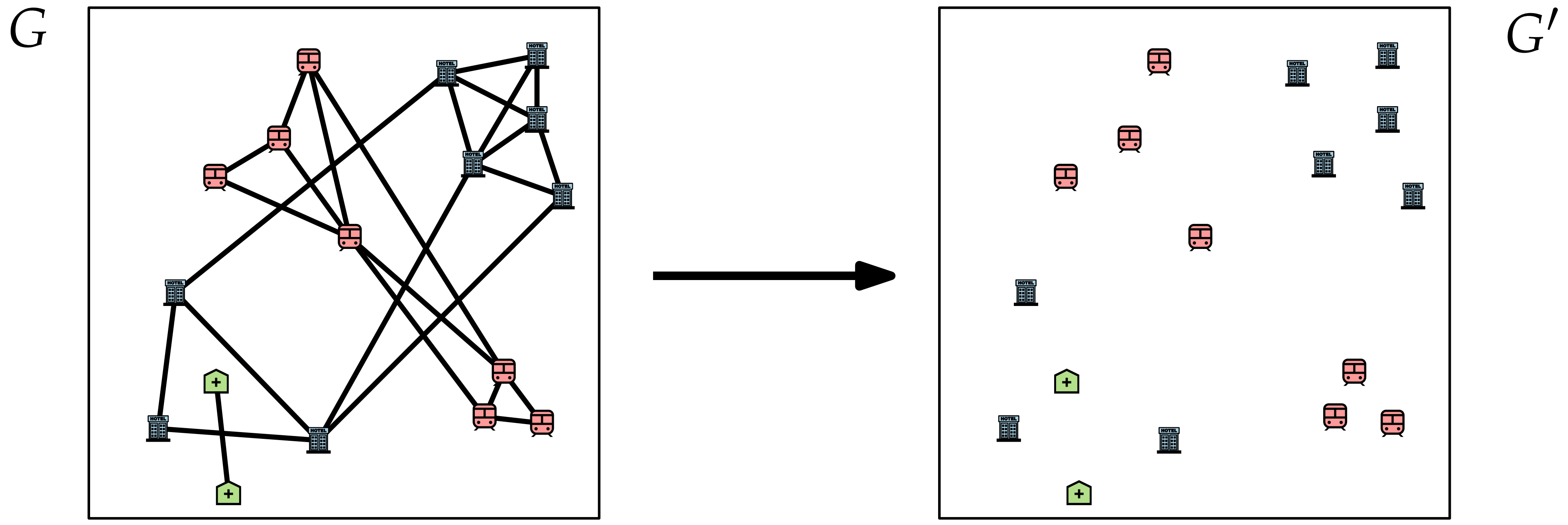

Pipeline
1.
2.
3.
4.
1. Proximity Graph
Delaunay Triangulation
Gabriel Graph
β-Skeleton
2. Planar Spanning Forest
3. Edge Augmentation
4. Rendering
Line Voronoi Diagram
Tree Representation
Polygon Representation

Planar Spanning Forest: Heuristics

Planar Spanning Forest: Heuristics



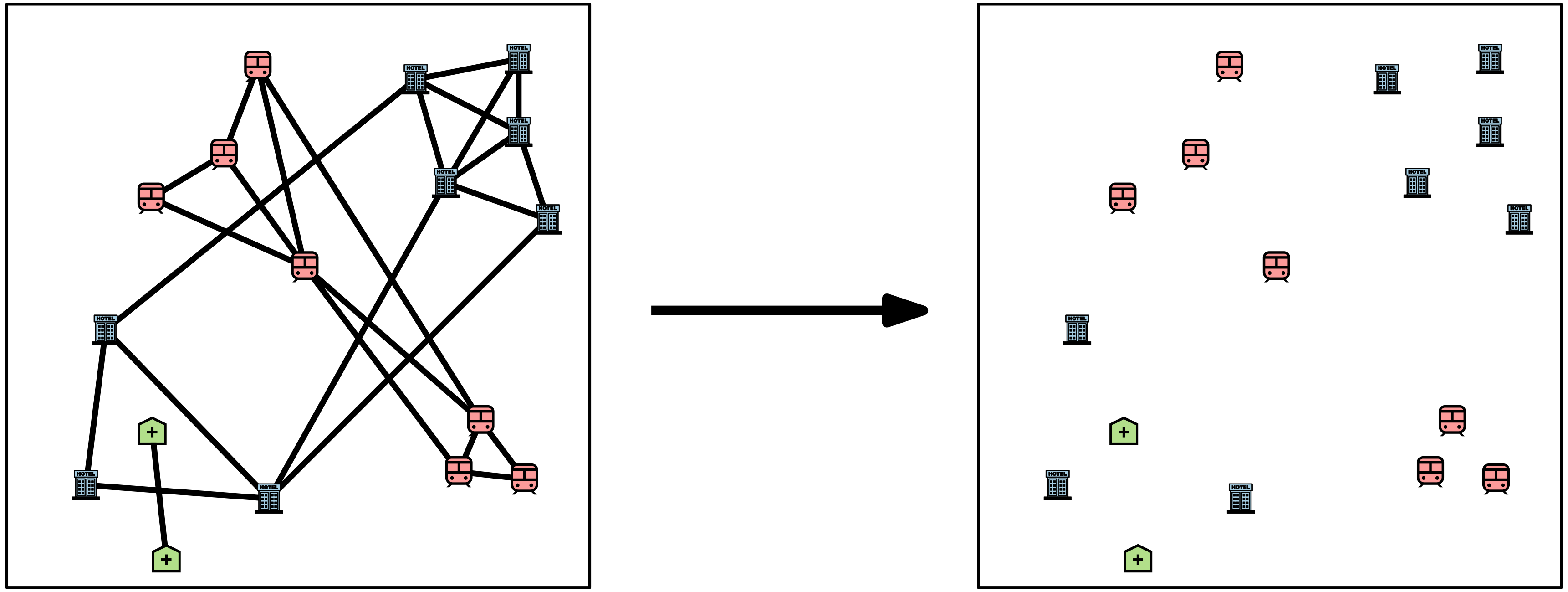
Planar Spanning Forest: Heuristics



Planar Spanning Forest: Heuristics

Greedy:

G



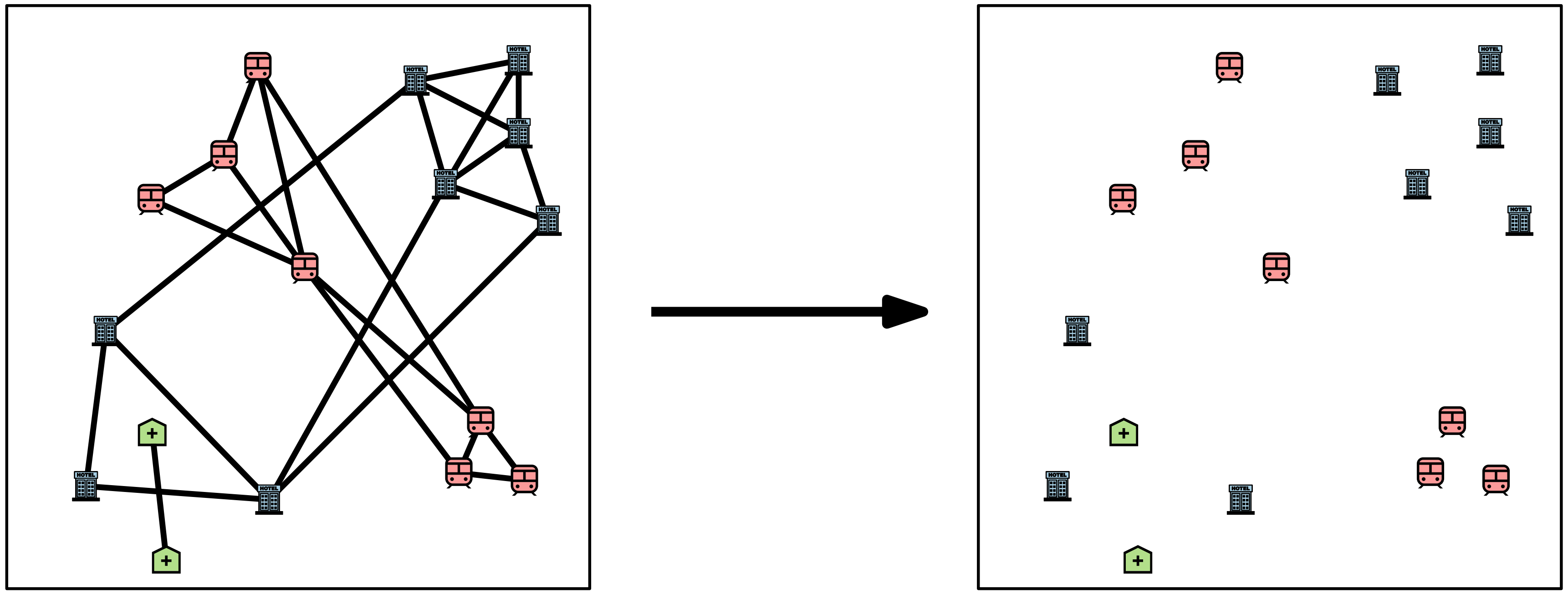
G'

Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.

G

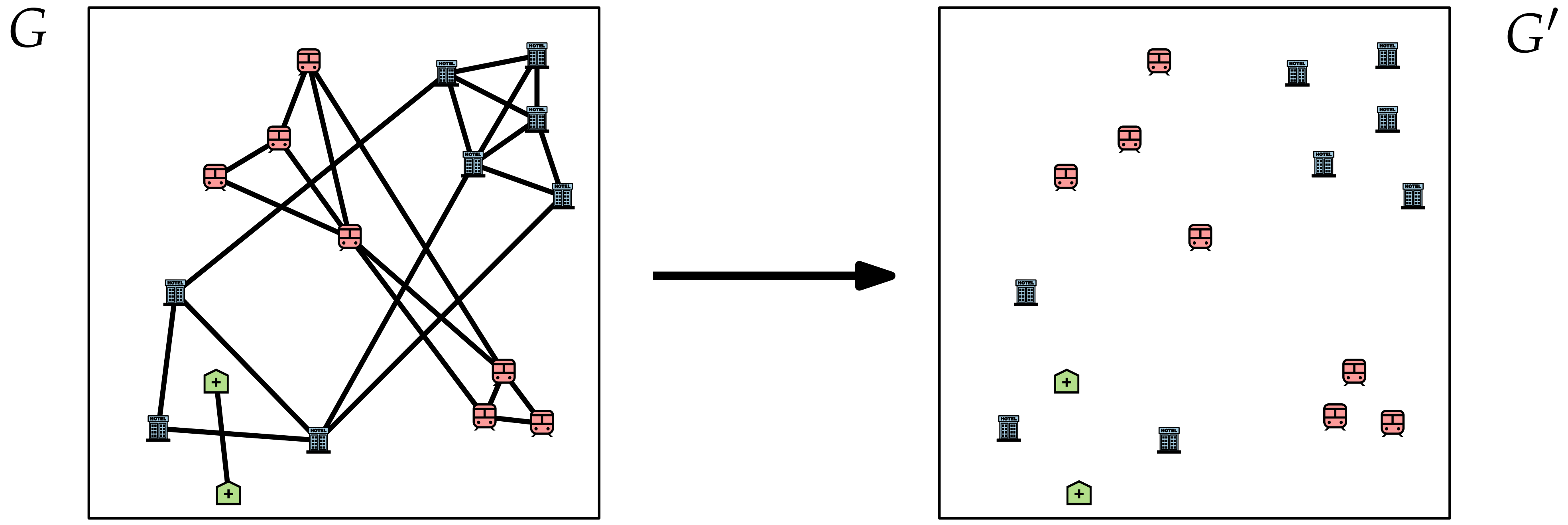


G'

Planar Spanning Forest: Heuristics

Greedy:

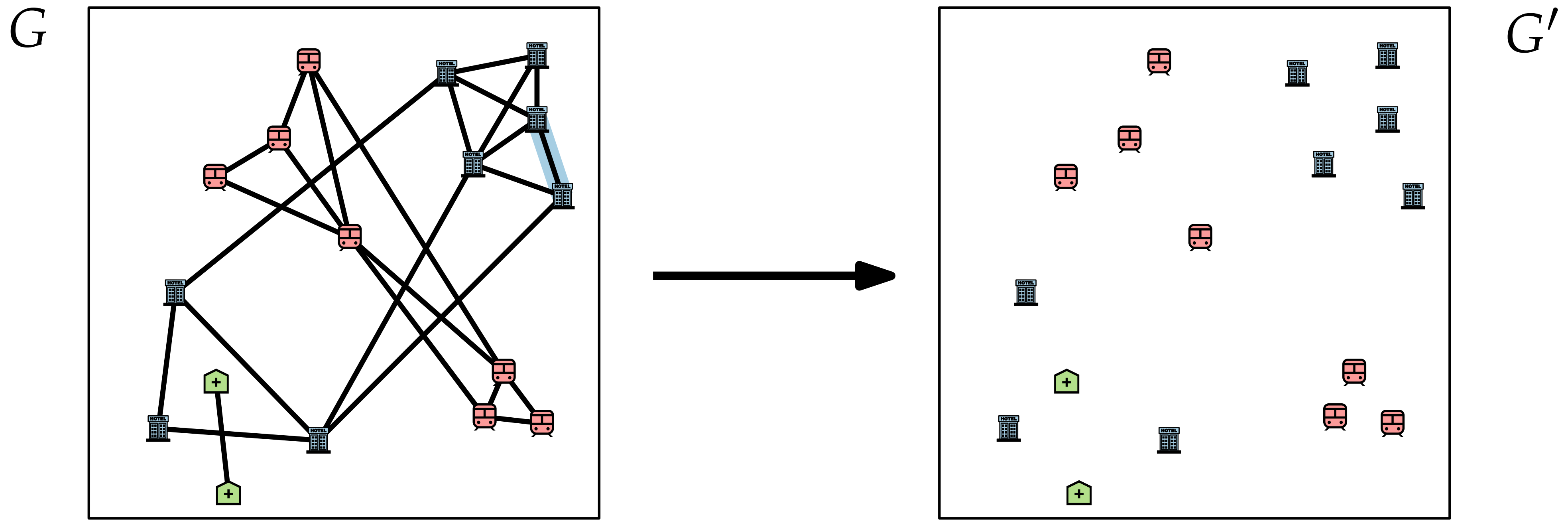
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.



Planar Spanning Forest: Heuristics

GREEDY:

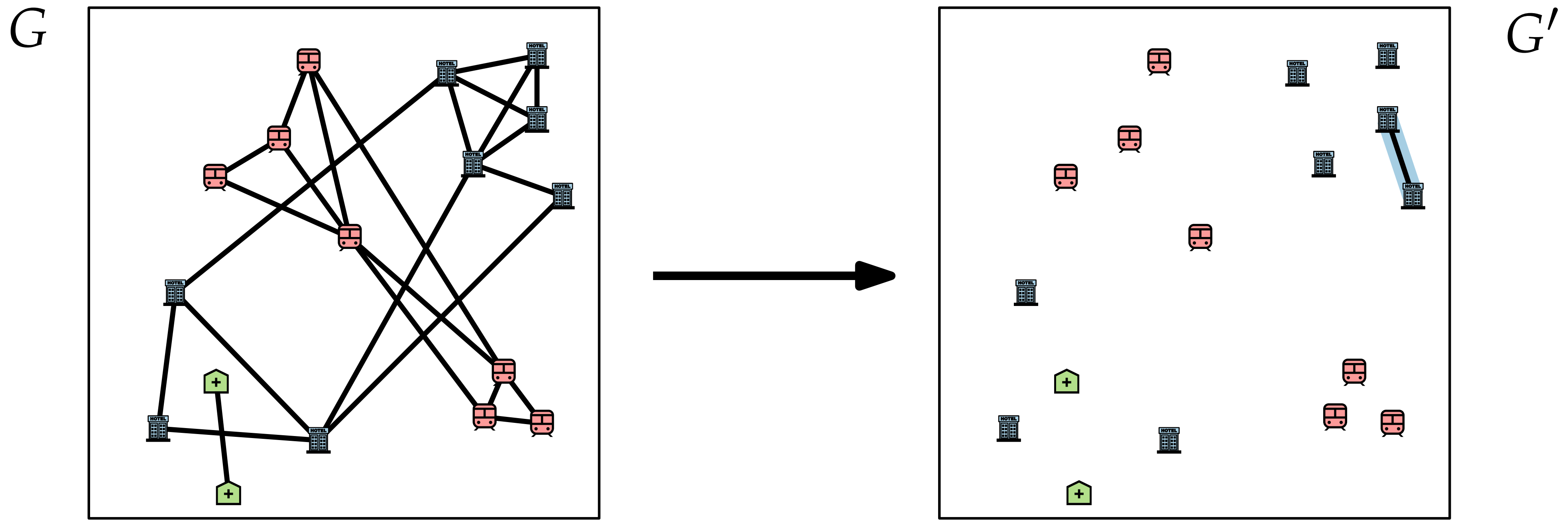
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.



Planar Spanning Forest: Heuristics

Greedy:

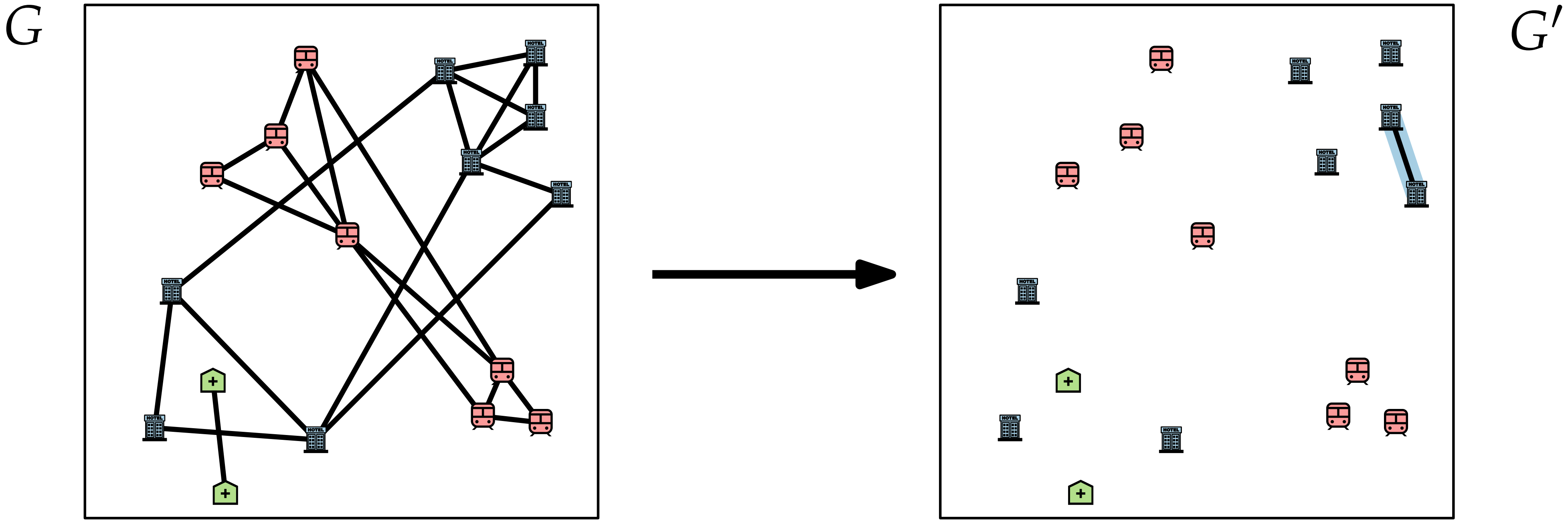
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.



Planar Spanning Forest: Heuristics

Greedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

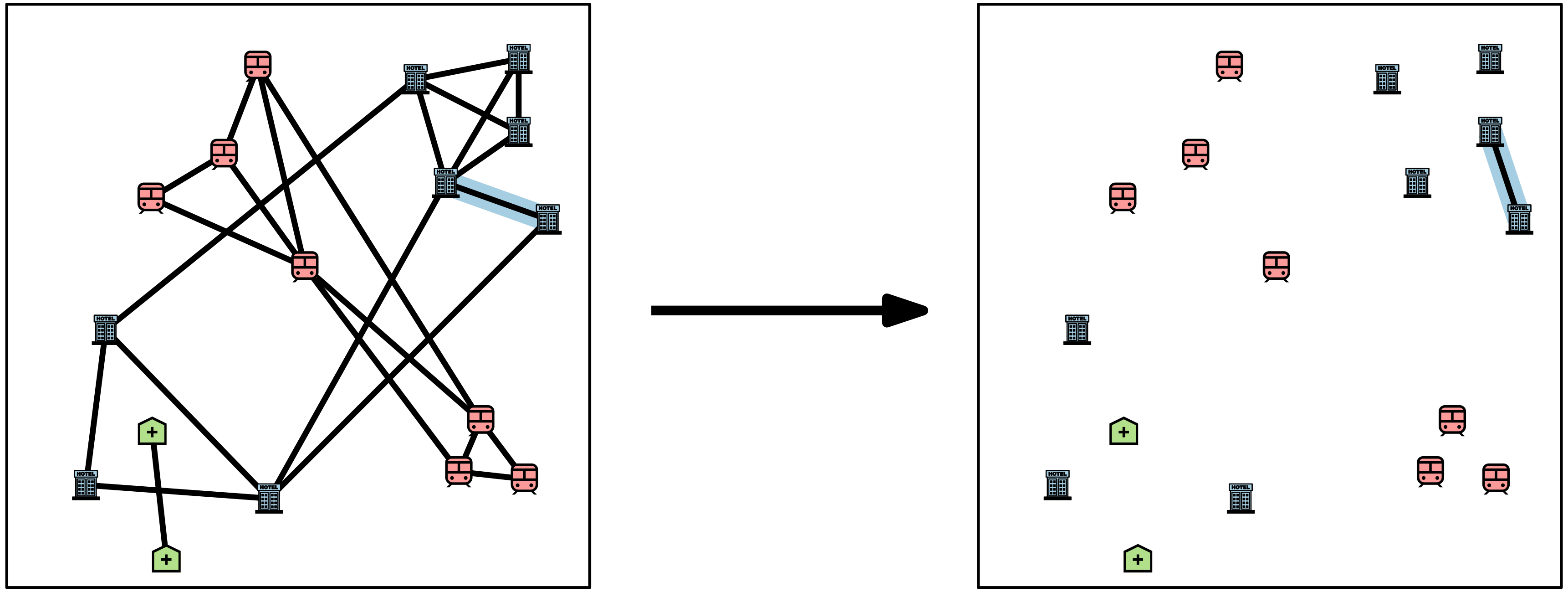


Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G

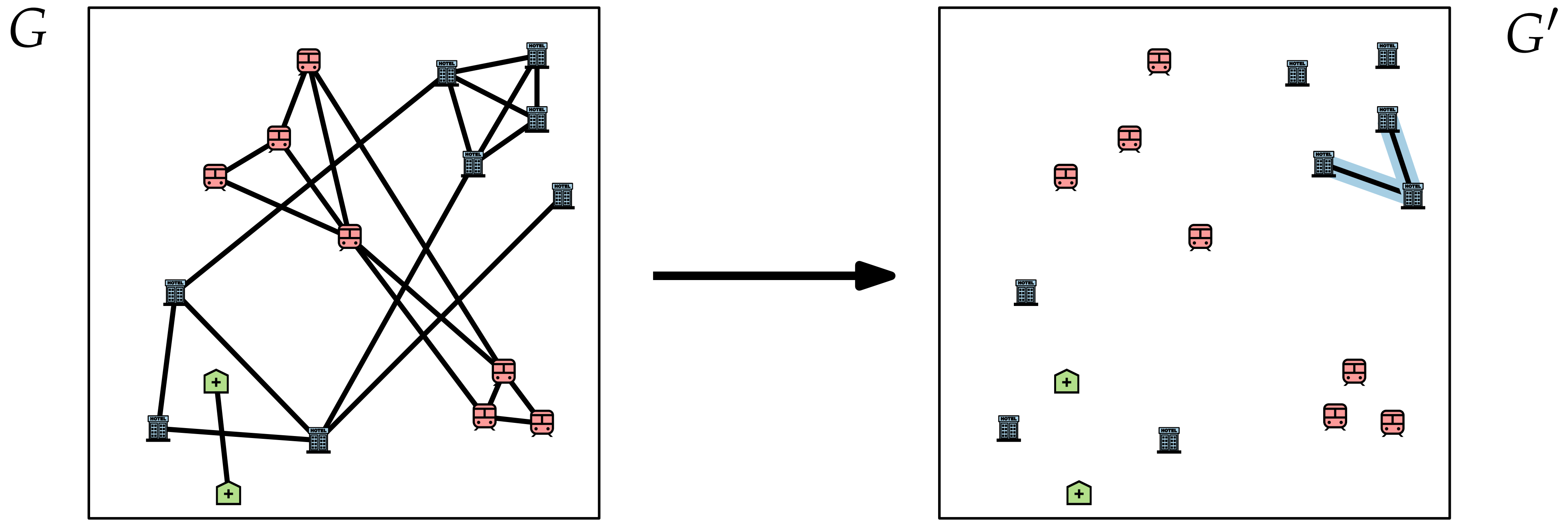


G'

Planar Spanning Forest: Heuristics

GREEDY:

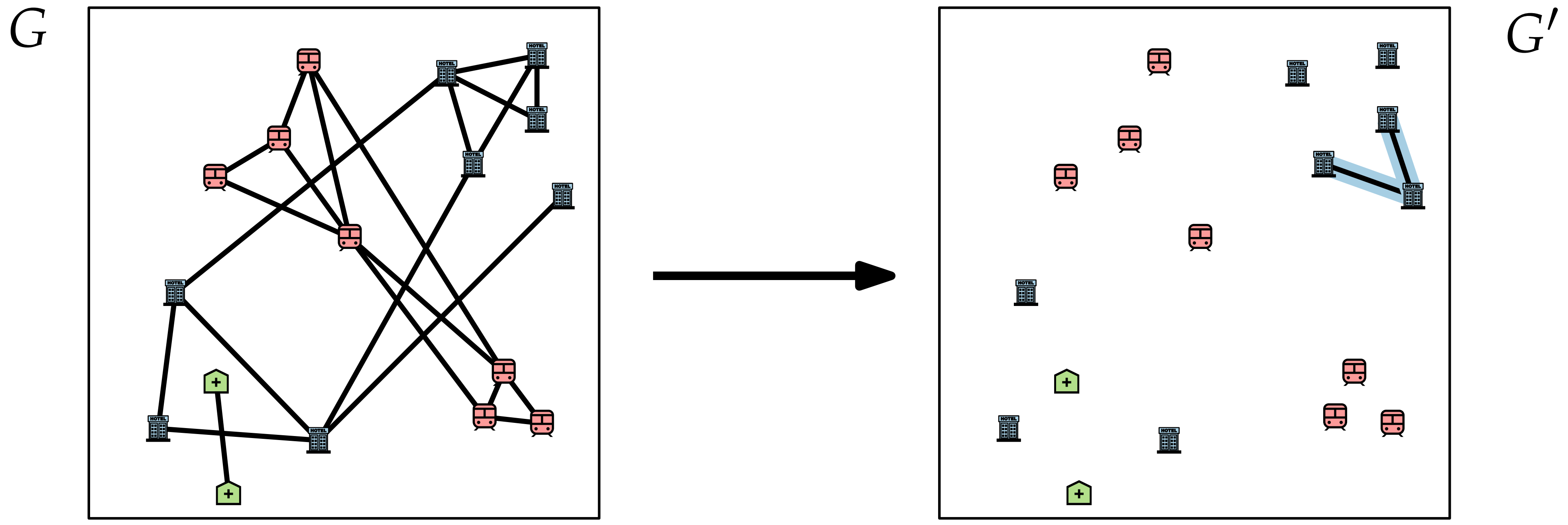
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

GREEDY:

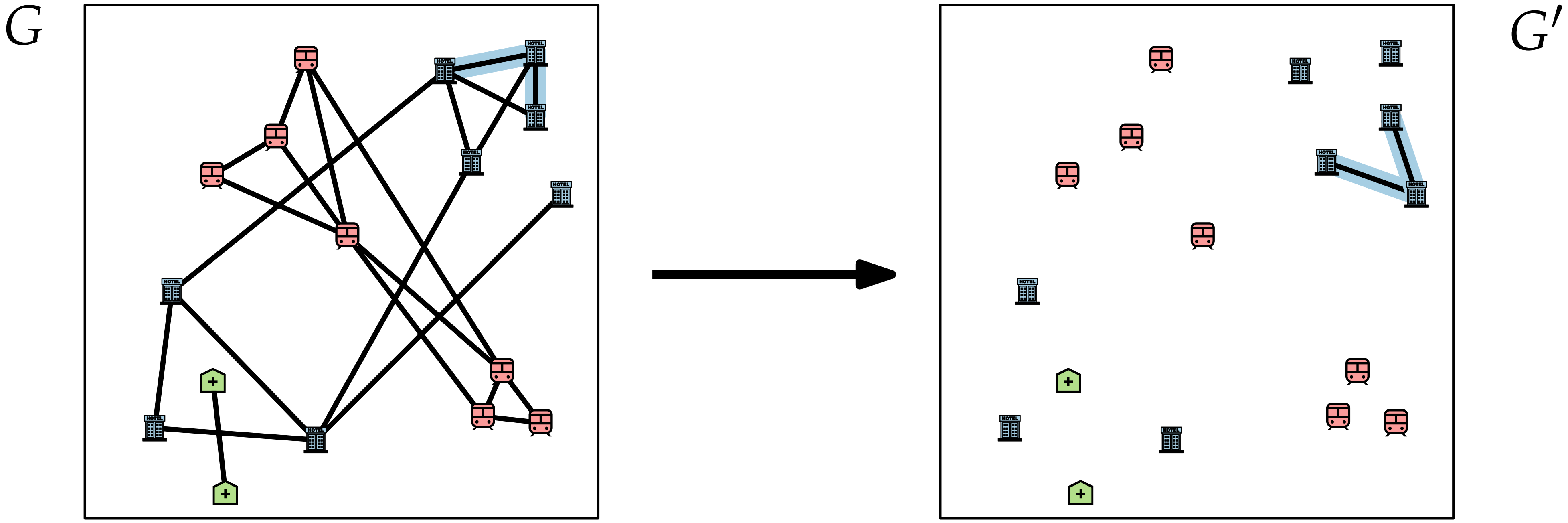
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

GREEDY:

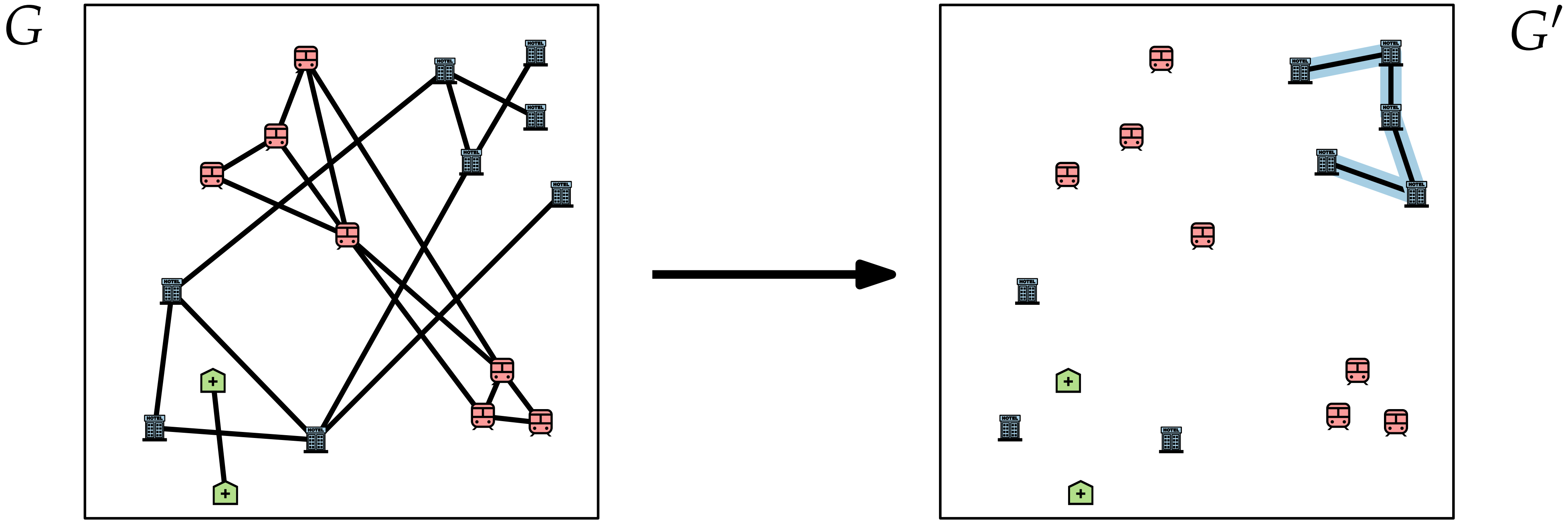
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

GREEDY:

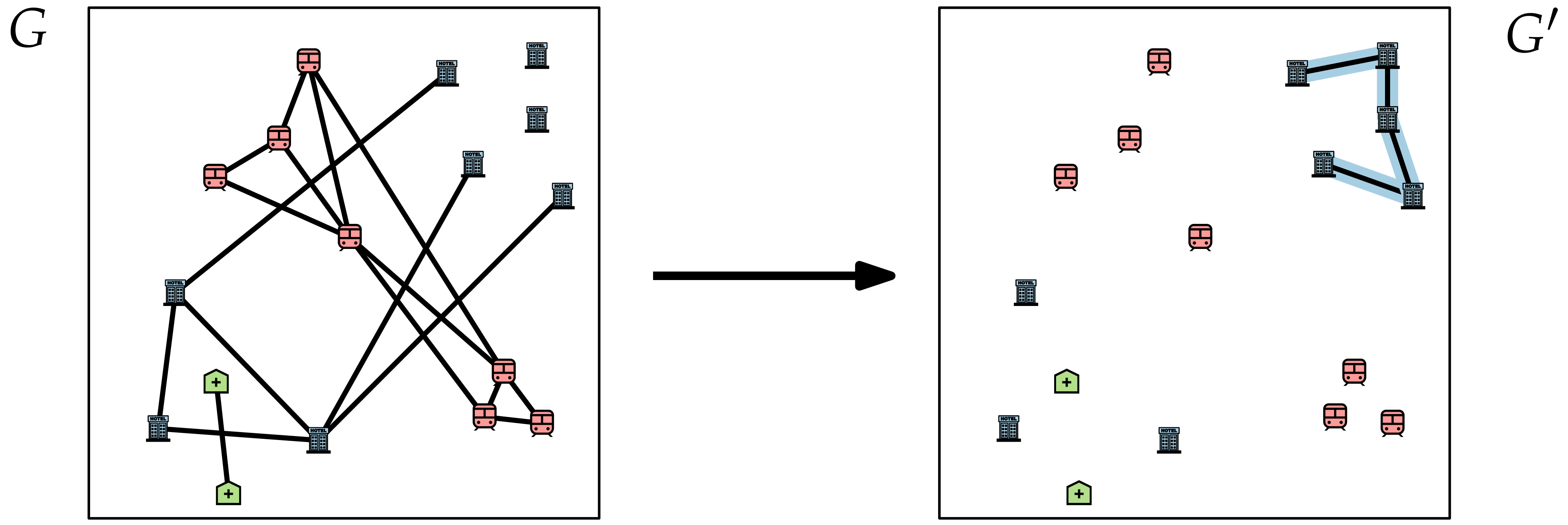
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

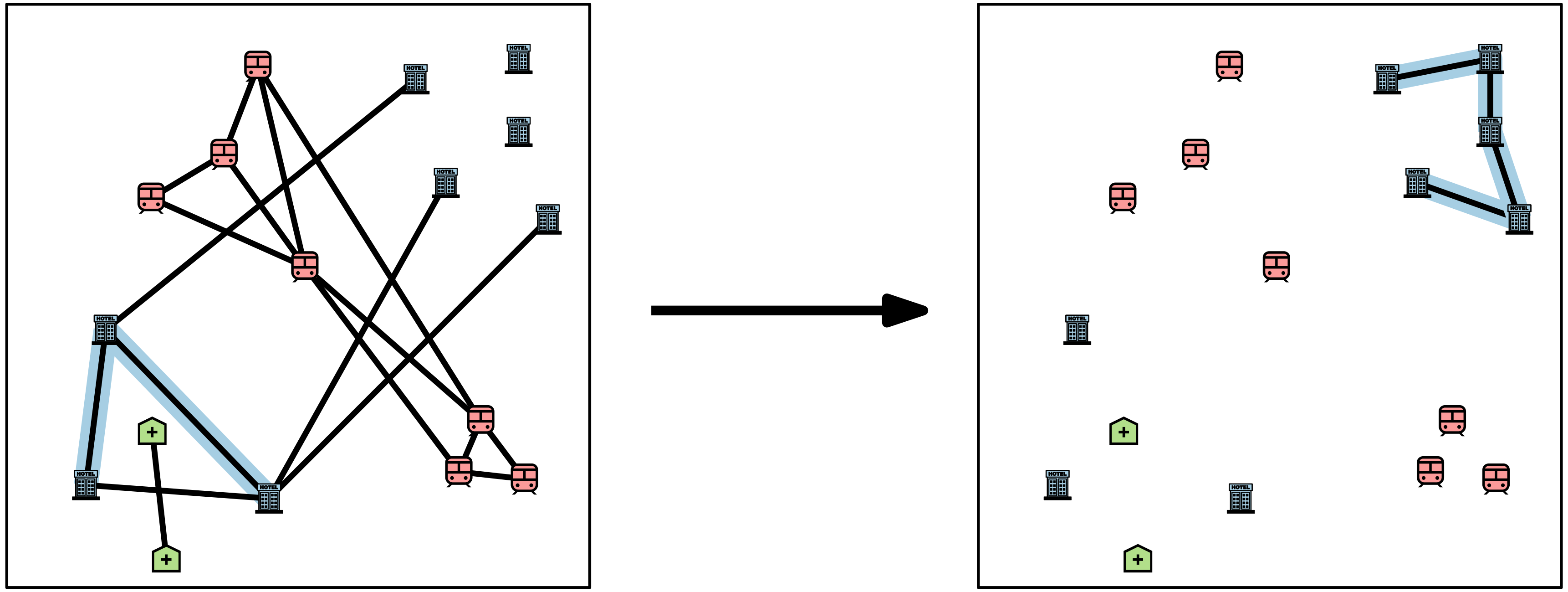


Planar Spanning Forest: Heuristics

Greedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G

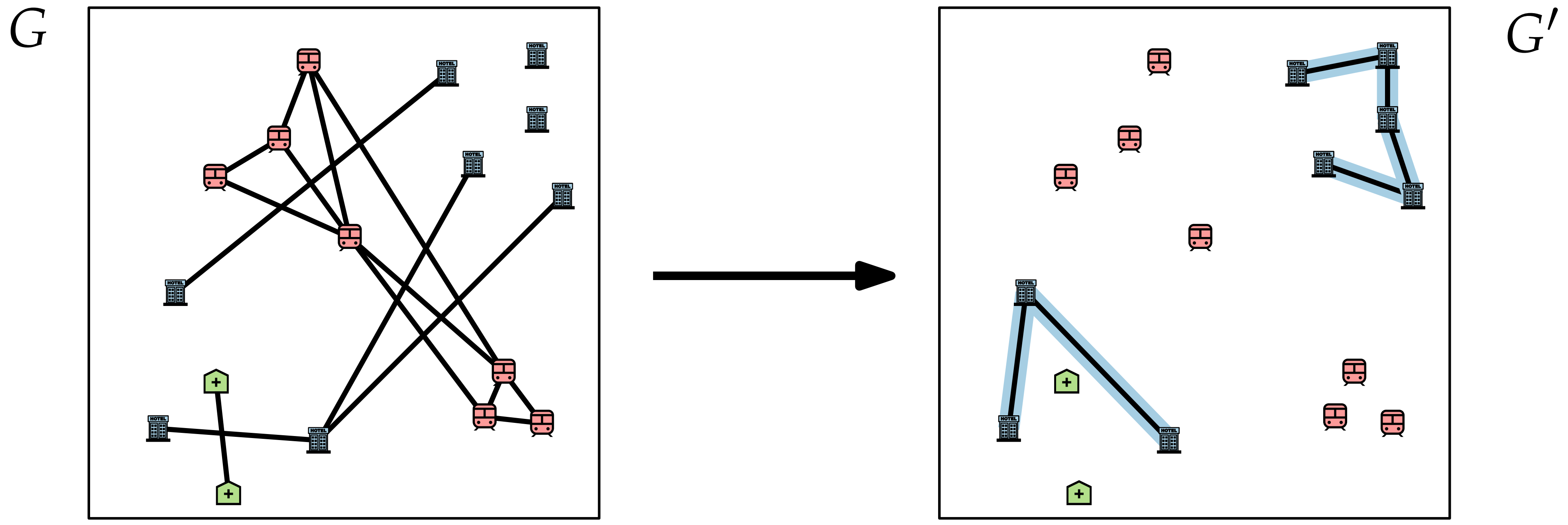


G'

Planar Spanning Forest: Heuristics

GREEDY:

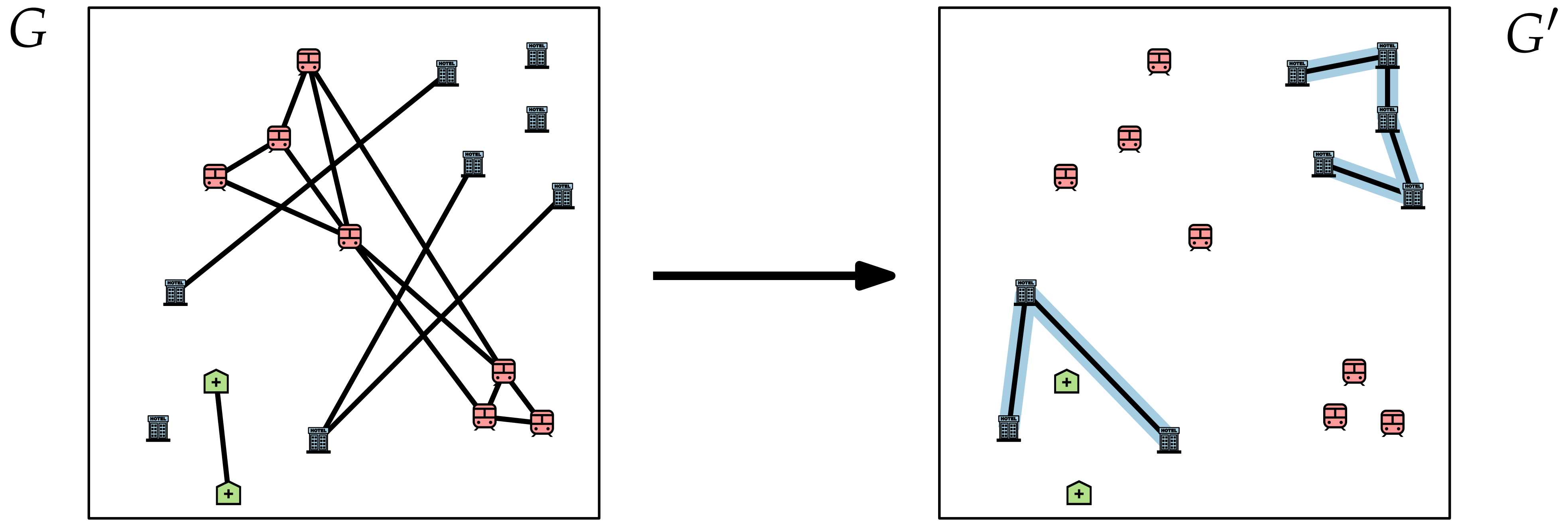
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

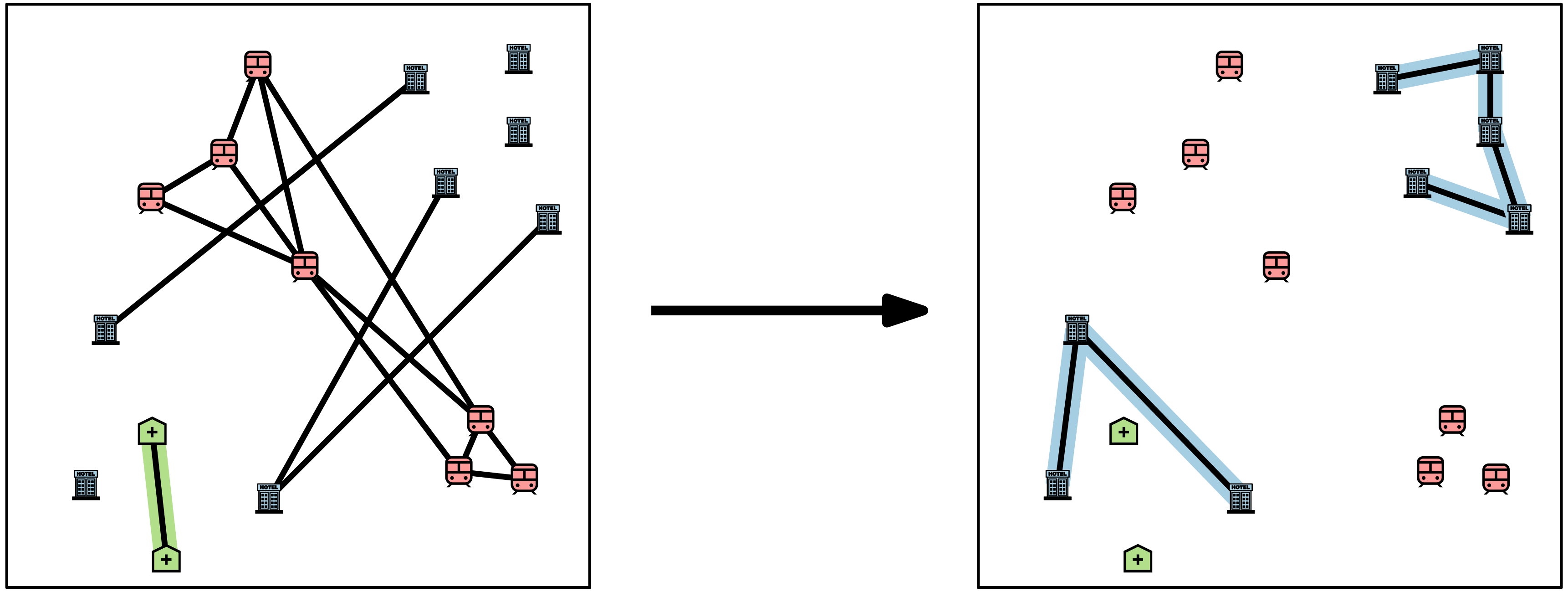


Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G

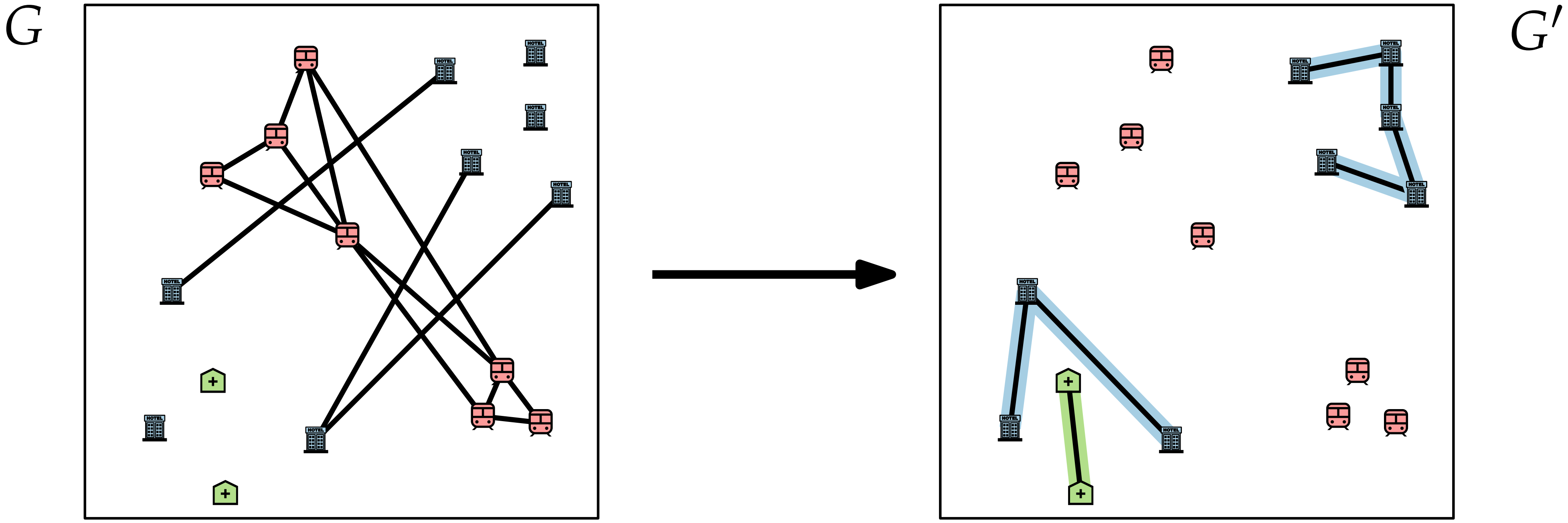


G'

Planar Spanning Forest: Heuristics

GREEDY:

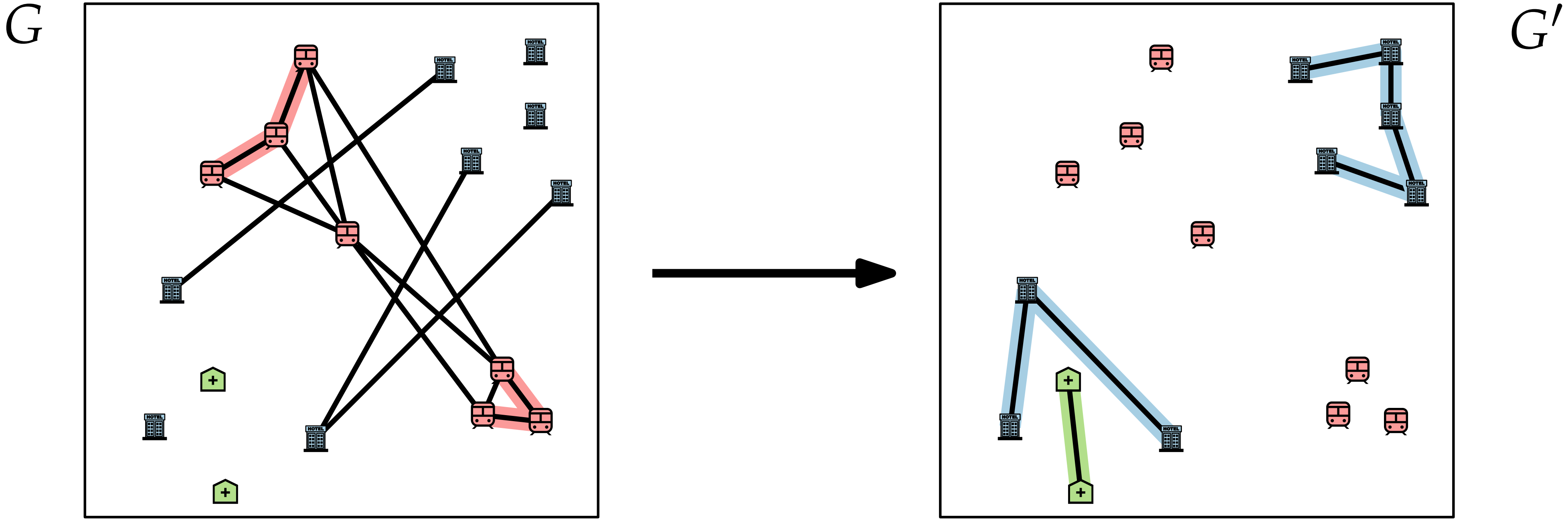
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

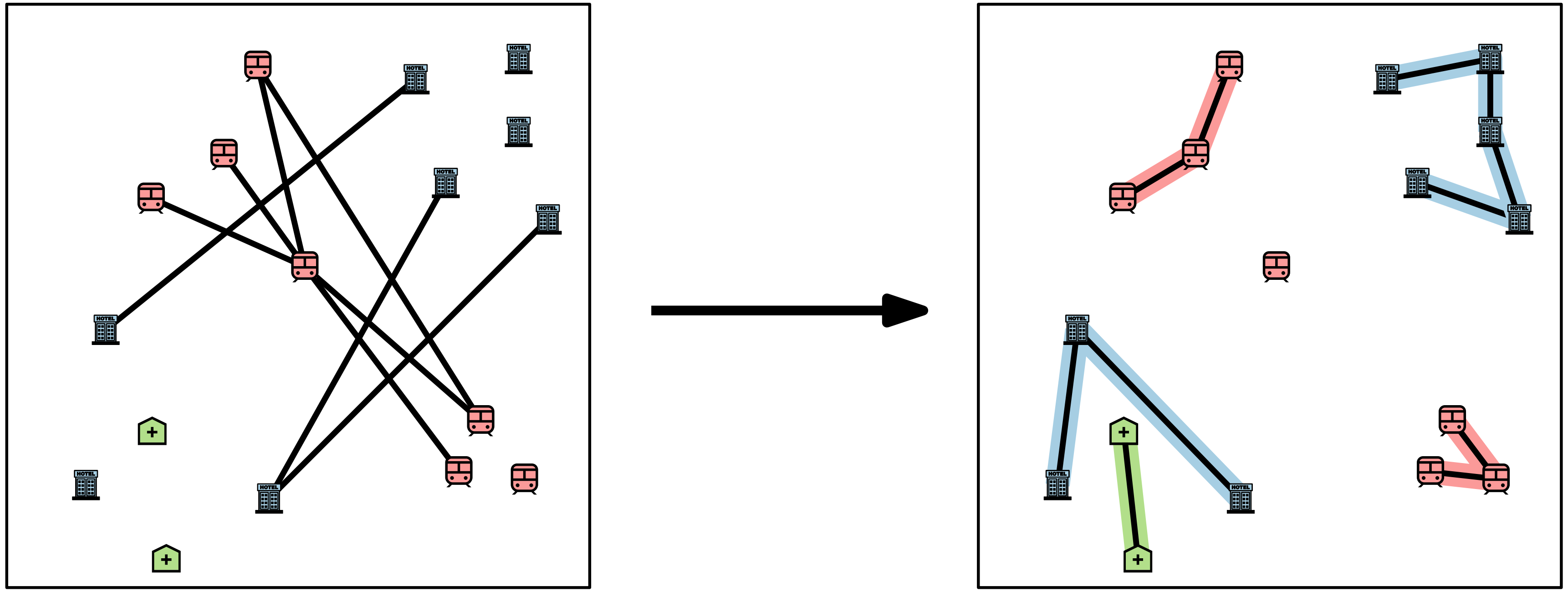


Planar Spanning Forest: Heuristics

Greedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G

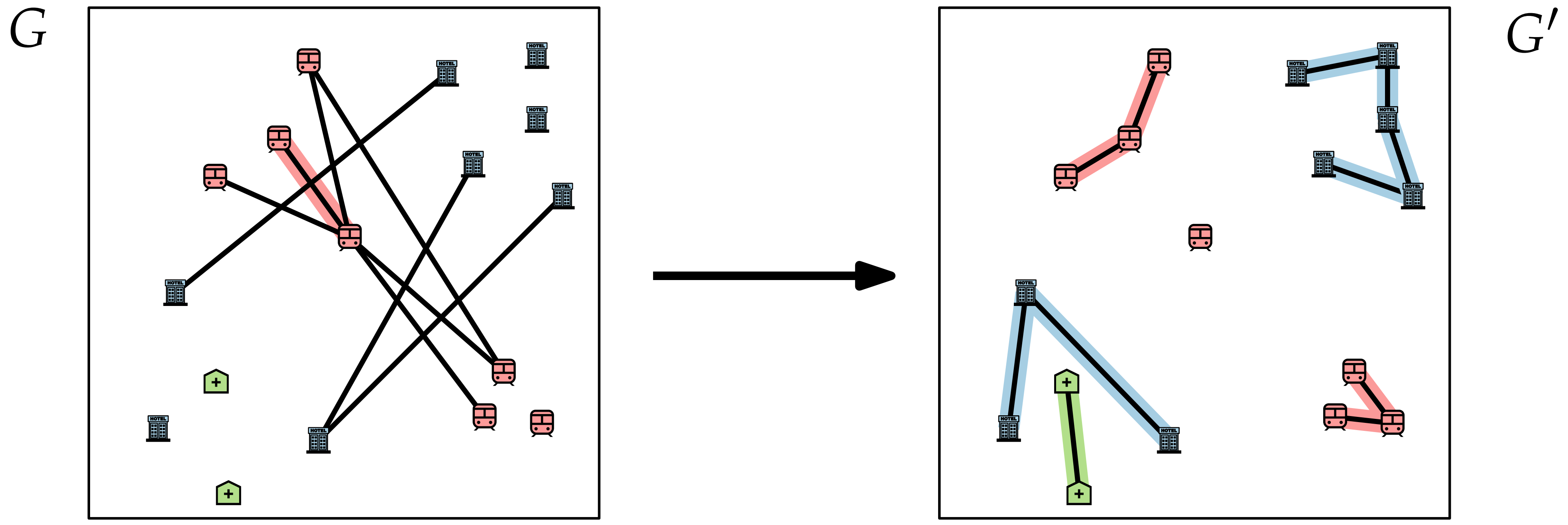


G'

Planar Spanning Forest: Heuristics

Greedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

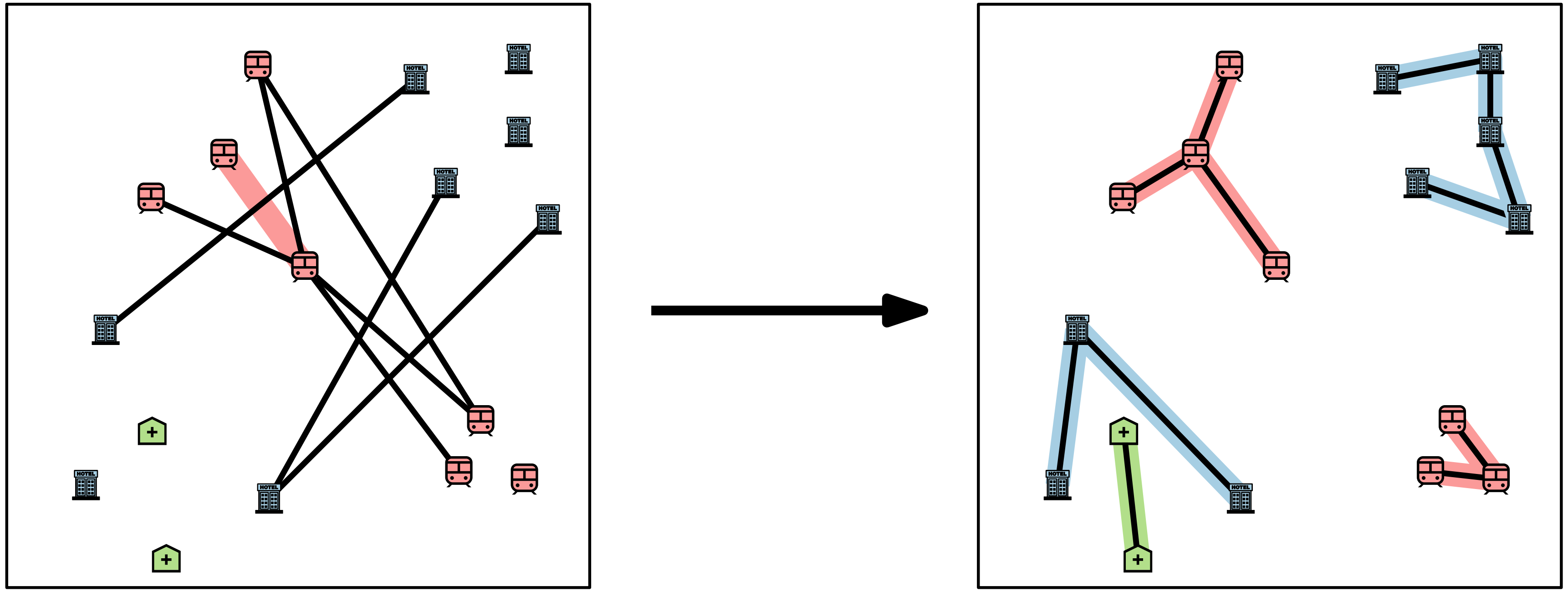


Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G



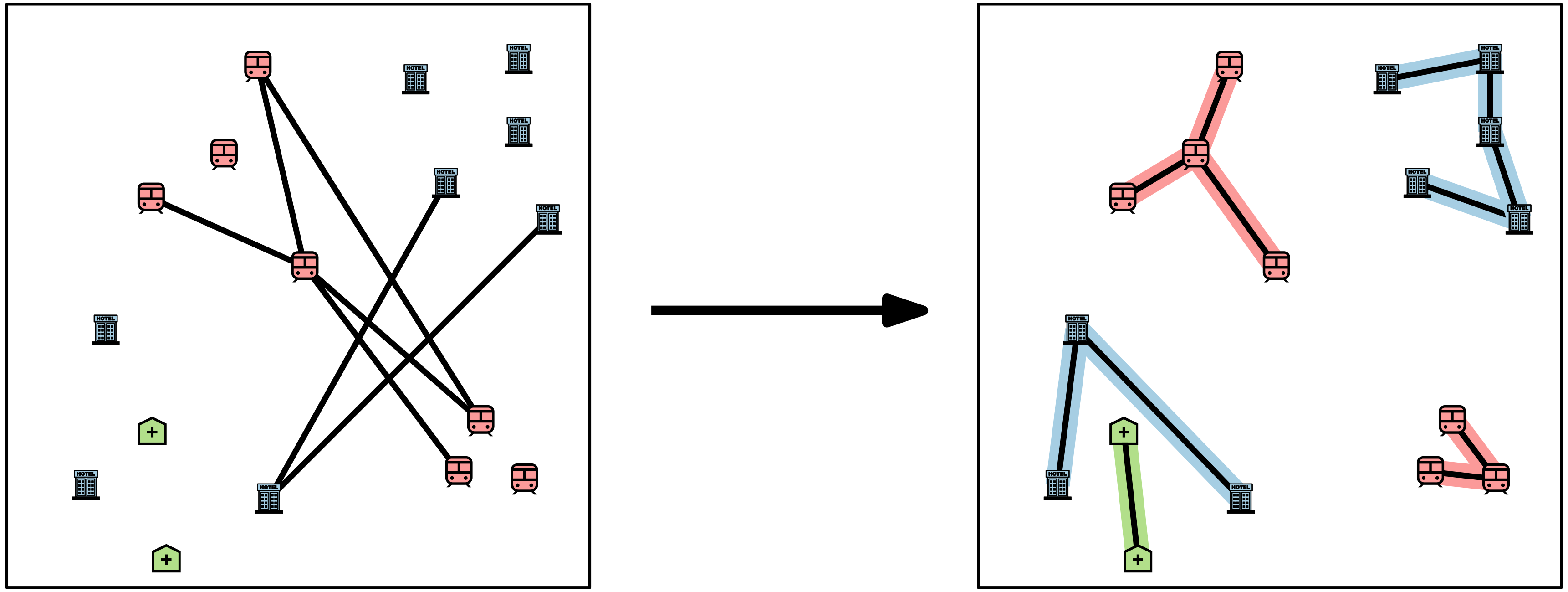
G'

Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G

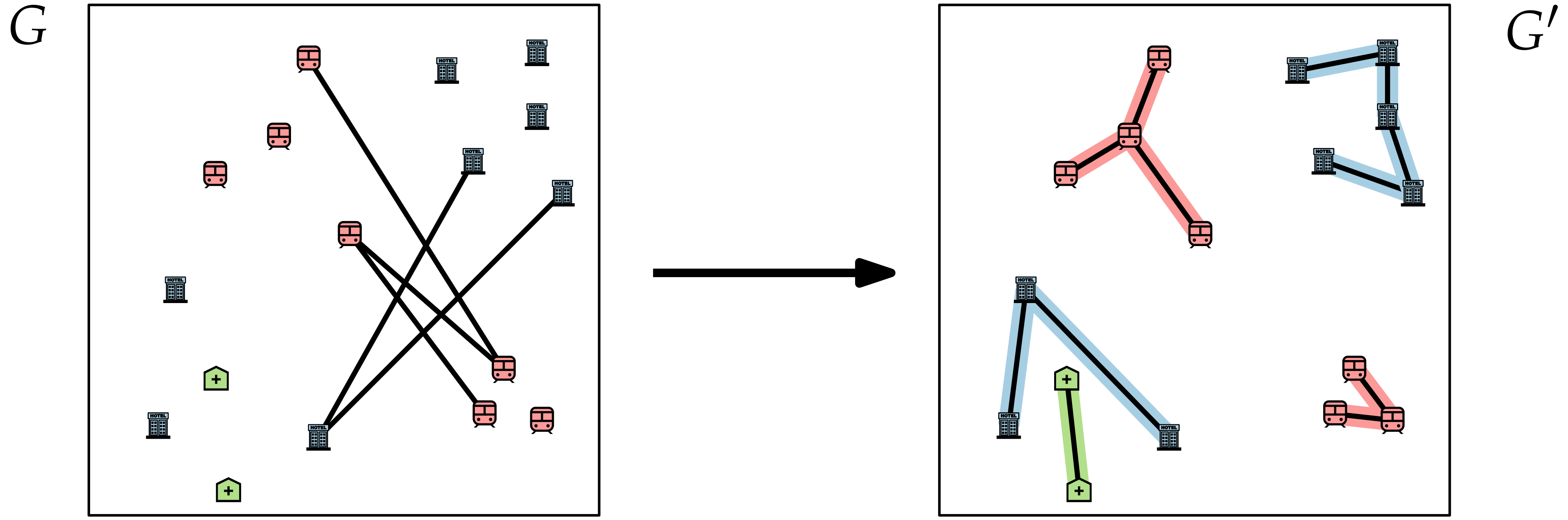


G'

Planar Spanning Forest: Heuristics

Greedy:

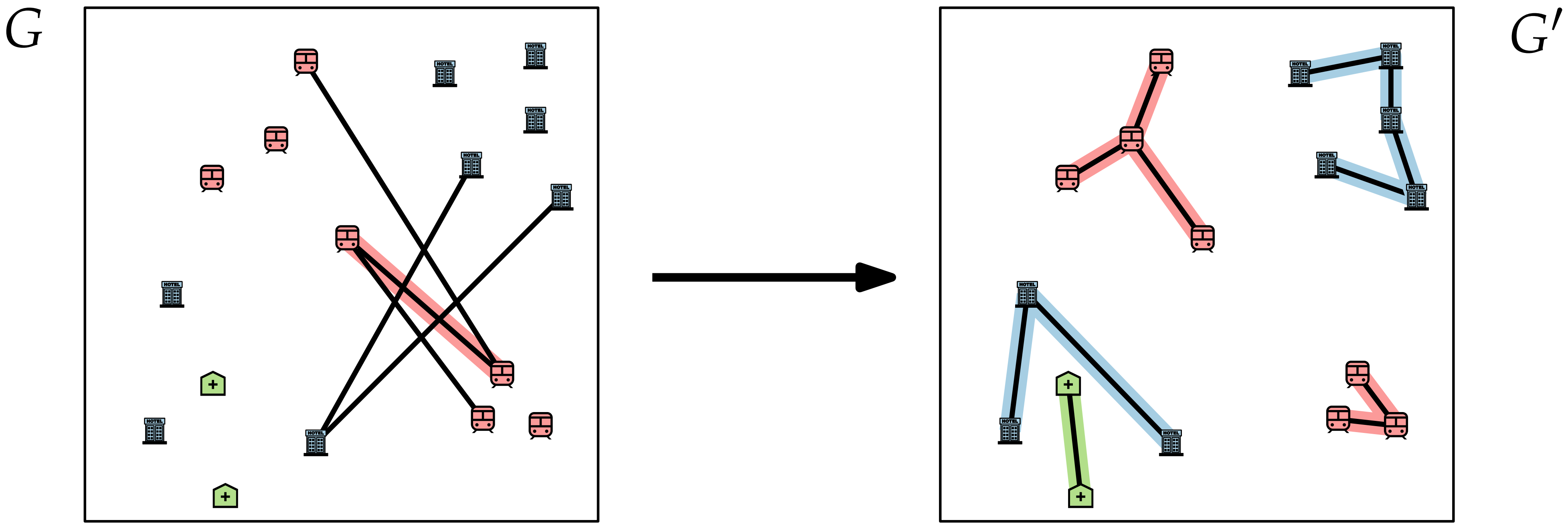
1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.



Planar Spanning Forest: Heuristics

Greedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

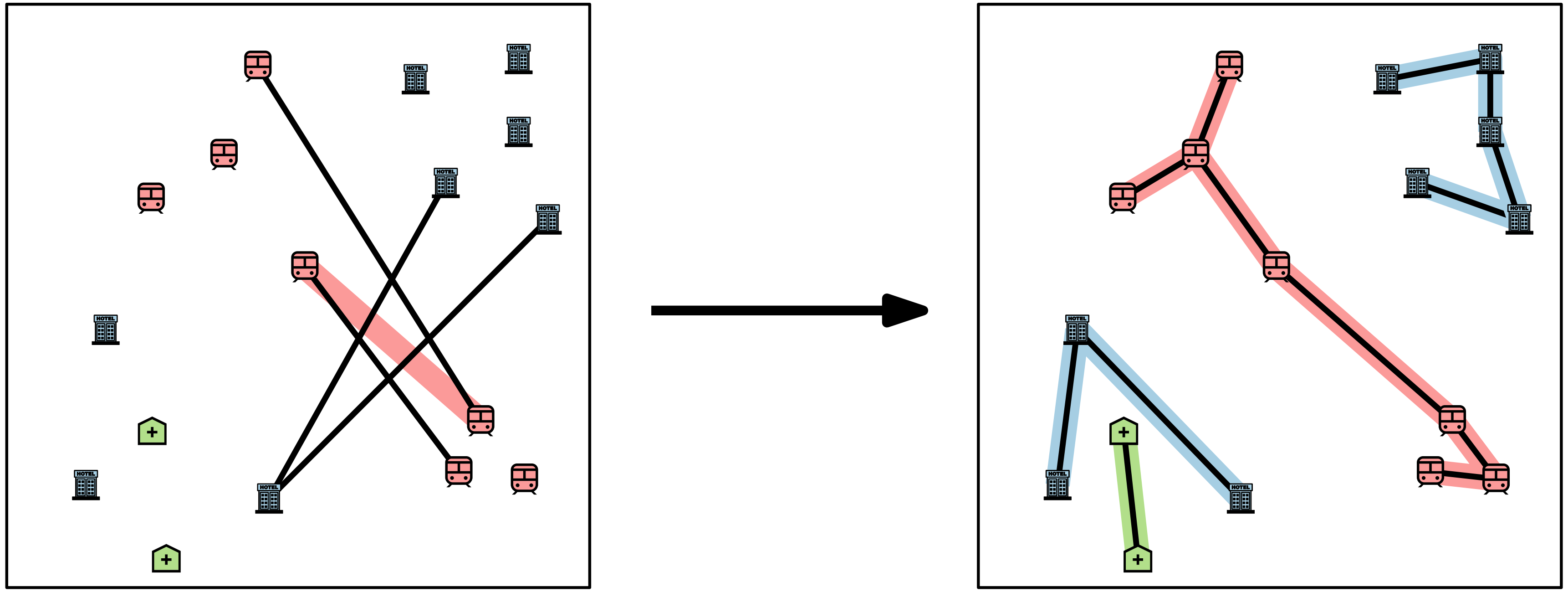


Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G



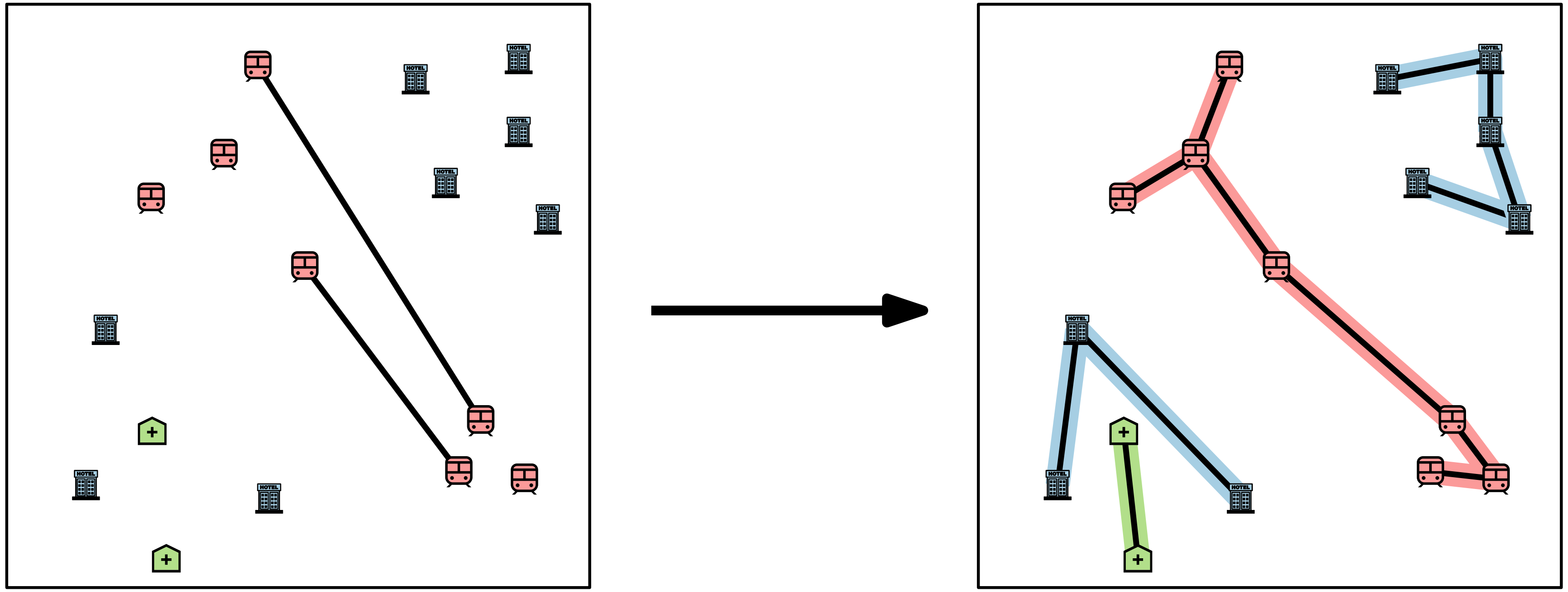
G'

Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

G

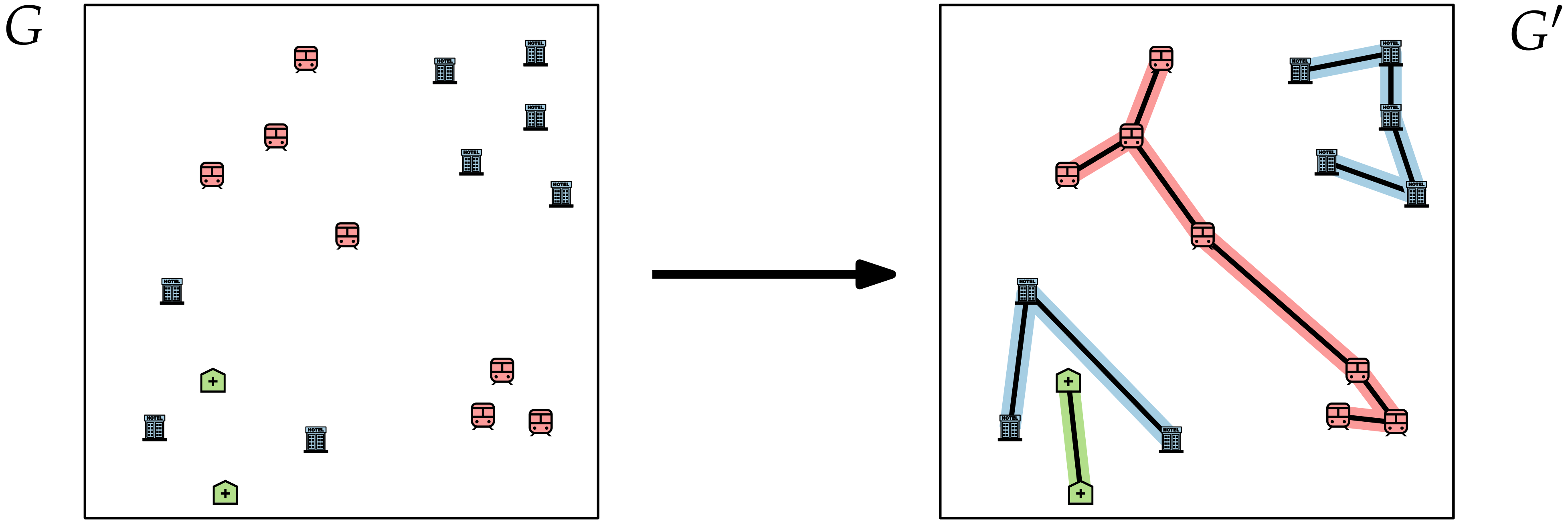


G'

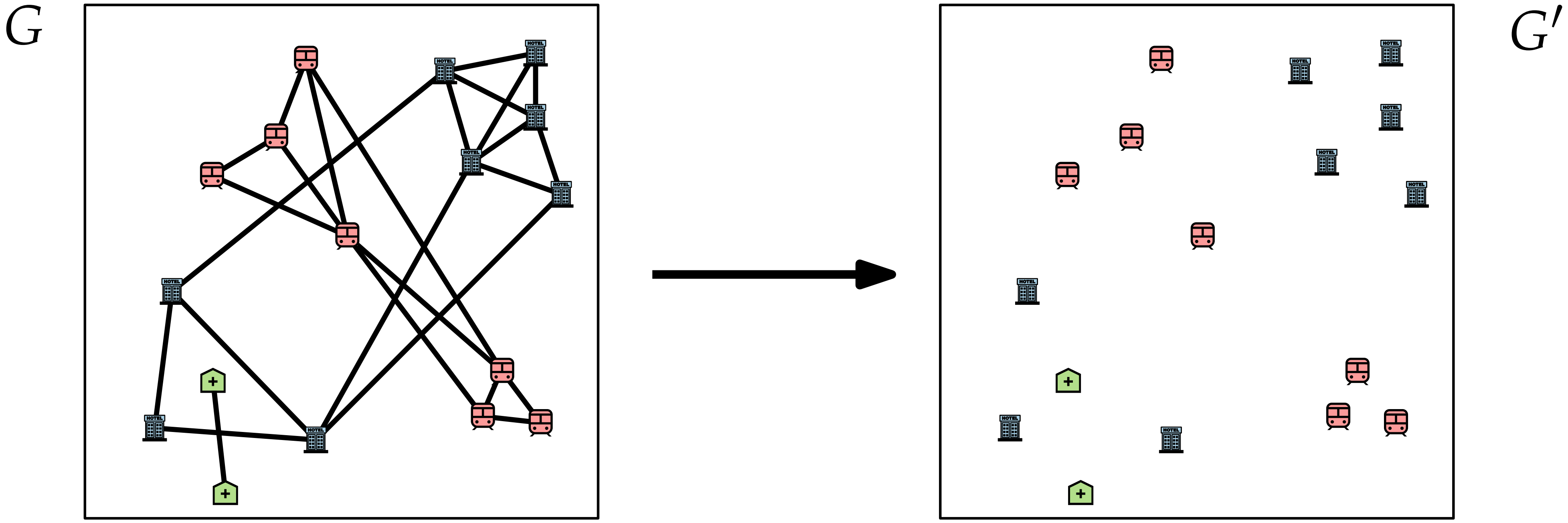
Planar Spanning Forest: Heuristics

GREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move edge e that is crossed by min. number of edges to G'.
3. Remove all edges that used to cross e.

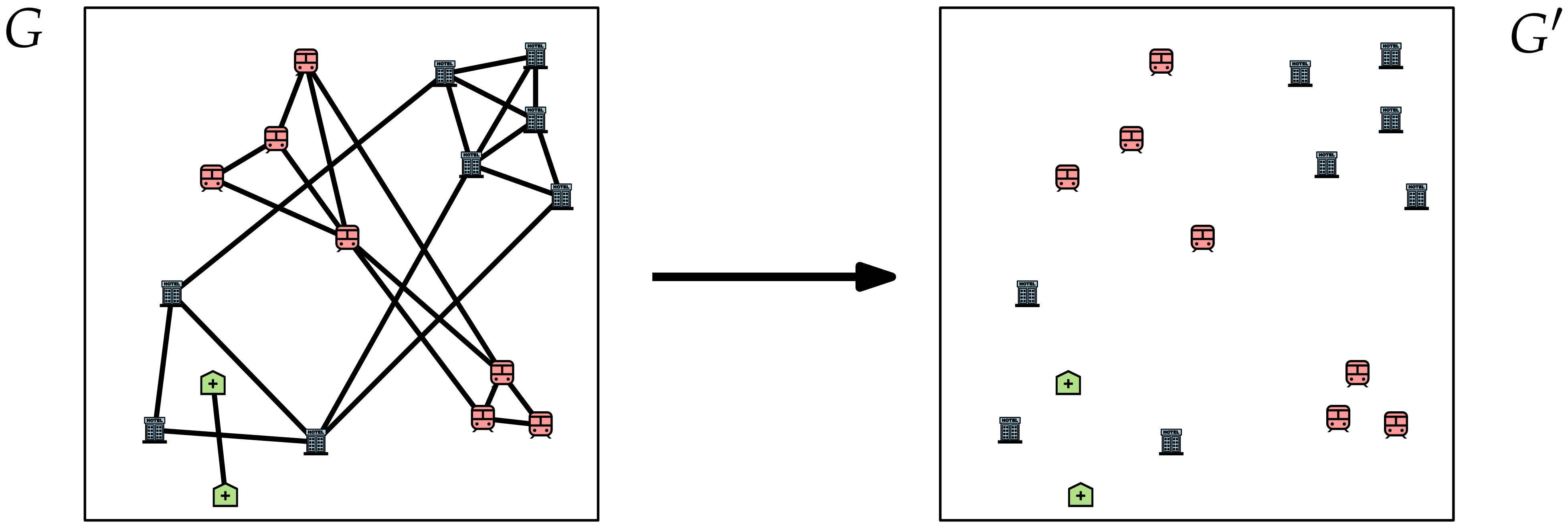


Planar Spanning Forest: Heuristics



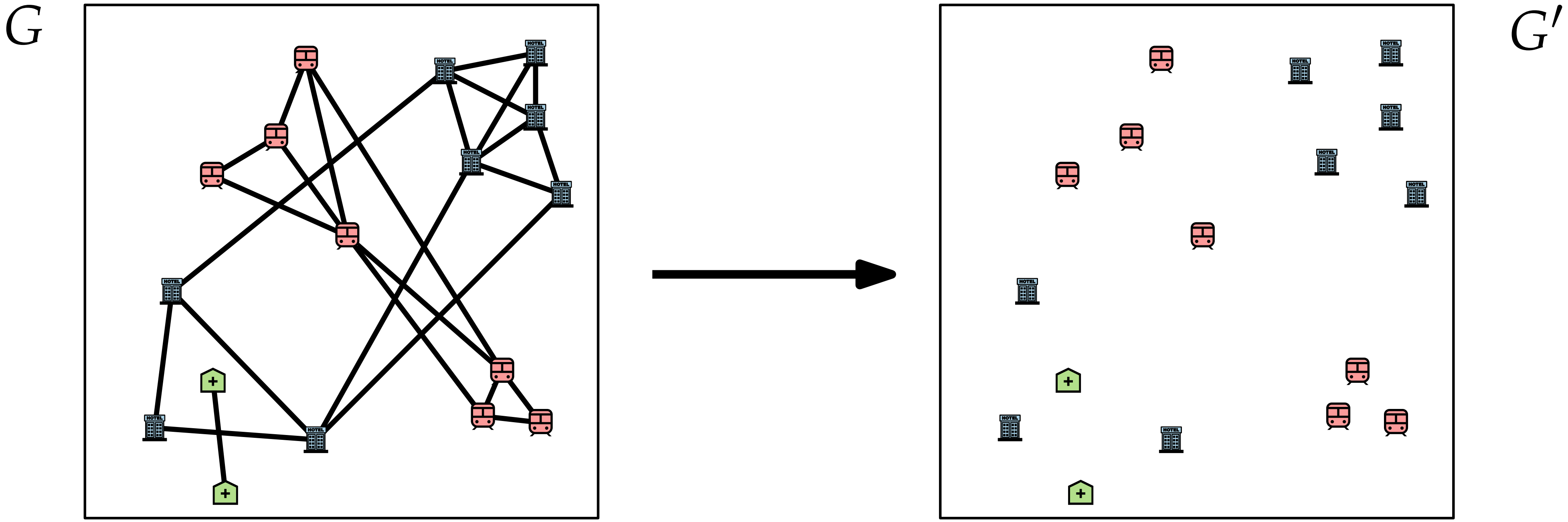
Planar Spanning Forest: Heuristics

REVERSEGREEDY:



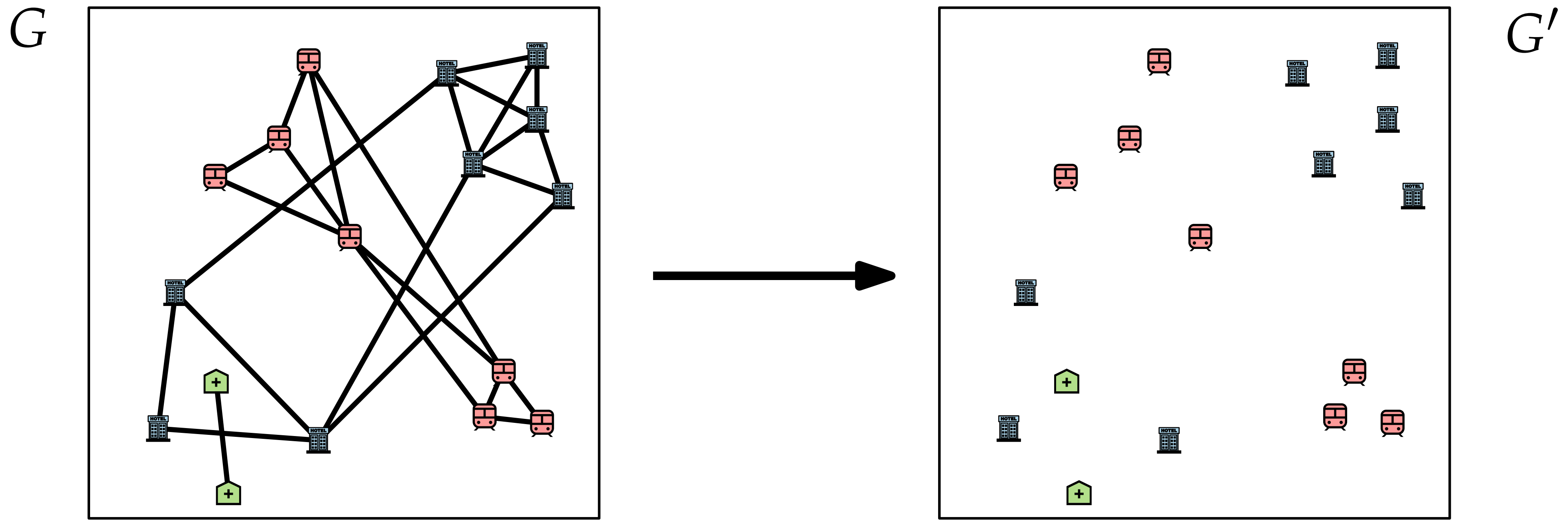
Planar Spanning Forest: Heuristics

ReverseGreedy: 1. Remove from G every edge that lies within a connected component of G'.



Planar Spanning Forest: Heuristics

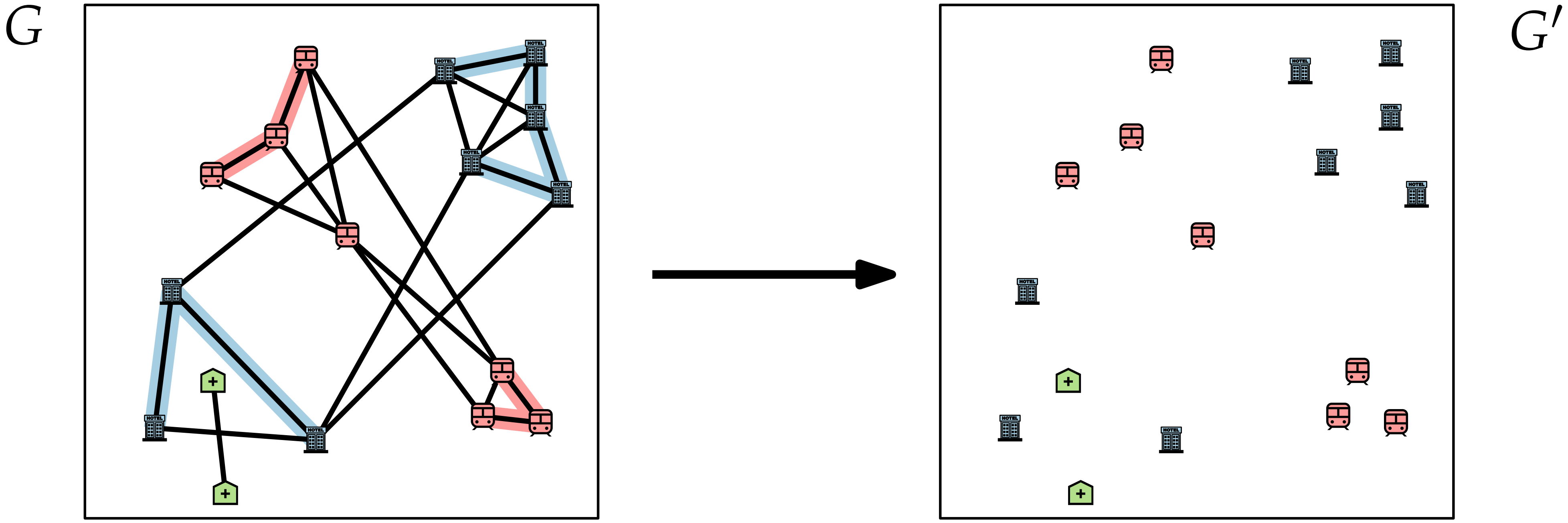
ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).



Planar Spanning Forest: Heuristics

ReverseGreedy:

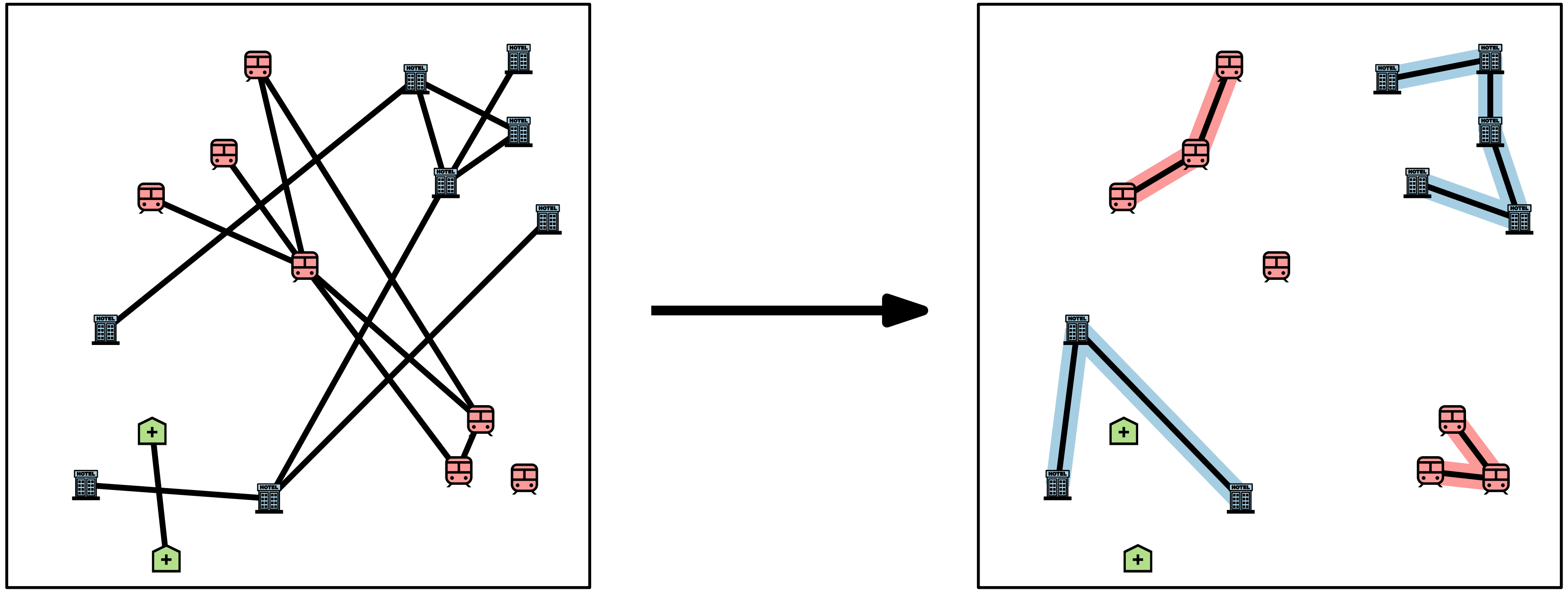
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).



Planar Spanning Forest: Heuristics

ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).

G

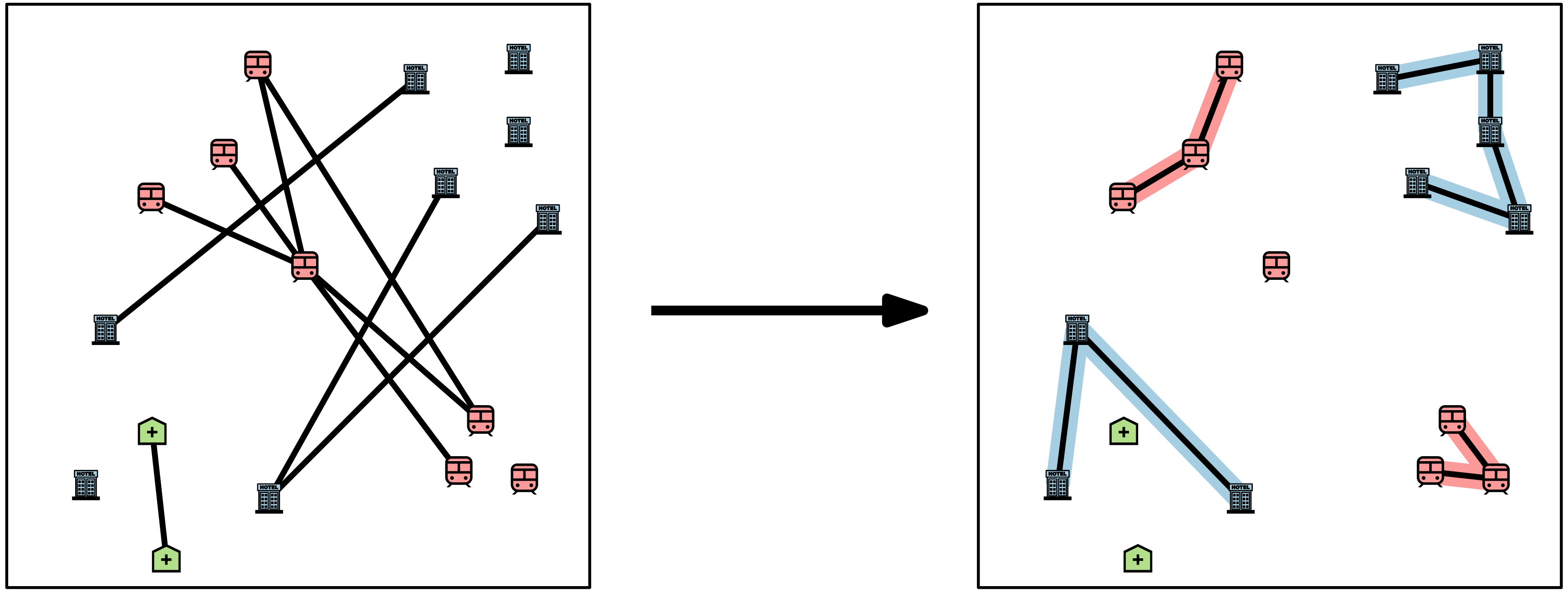


G'

Planar Spanning Forest: Heuristics

ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).

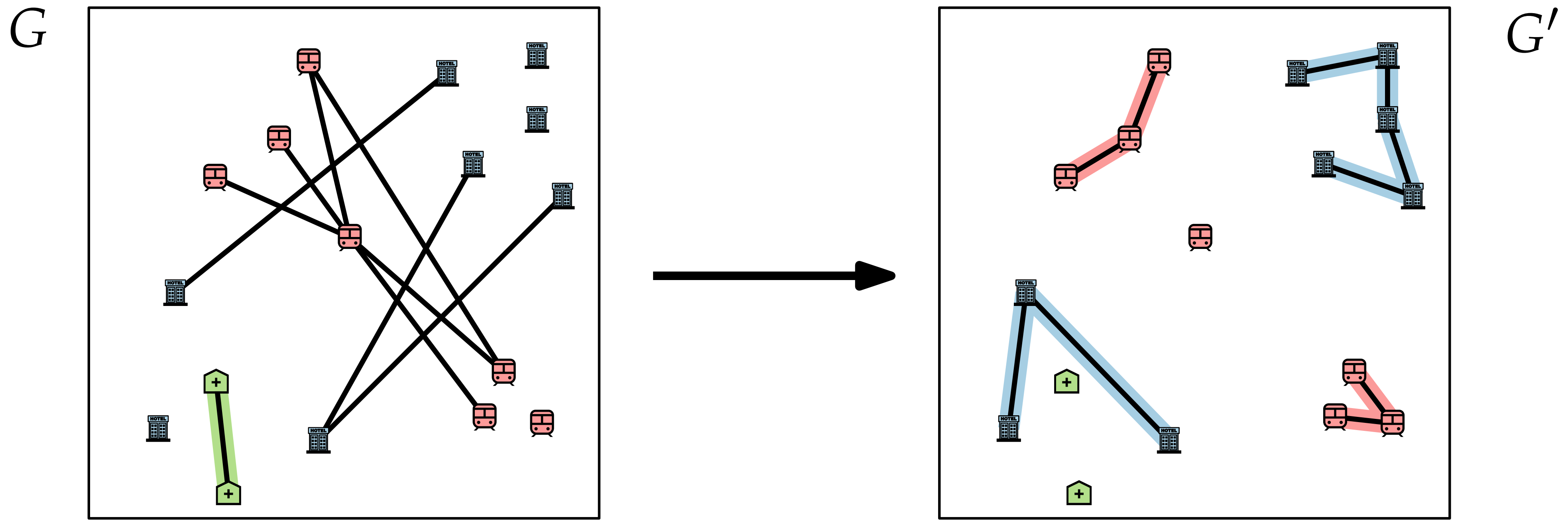
G



G'

Planar Spanning Forest: Heuristics

REVERSEGREEDY:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).

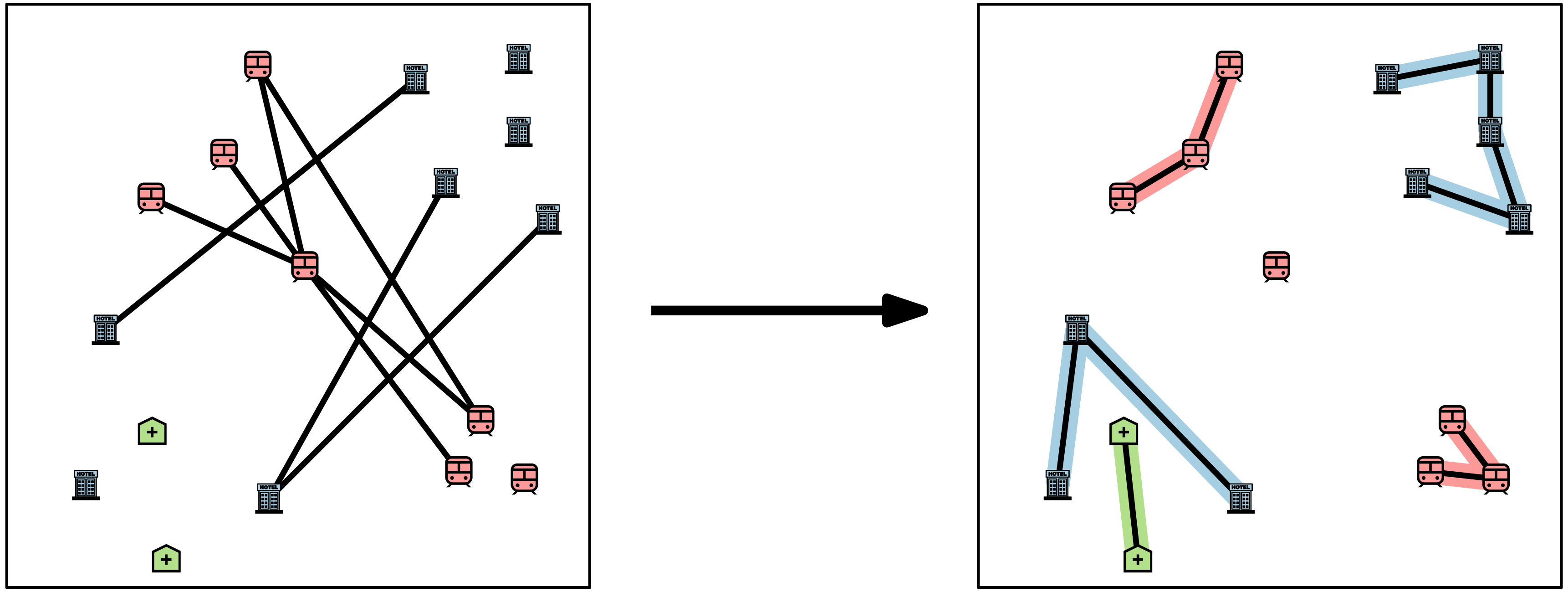


Planar Spanning Forest: Heuristics

ReverseGreedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).

G



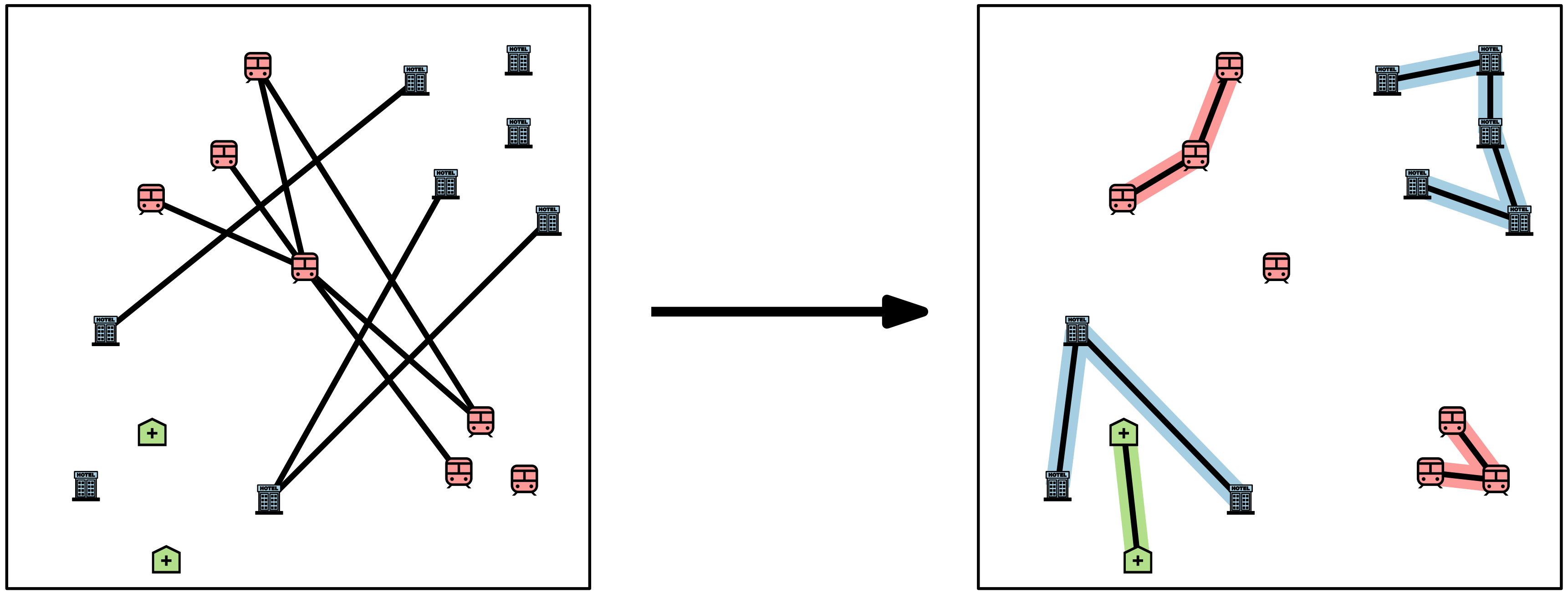
G'

Planar Spanning Forest: Heuristics

REVERSEGREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges

G

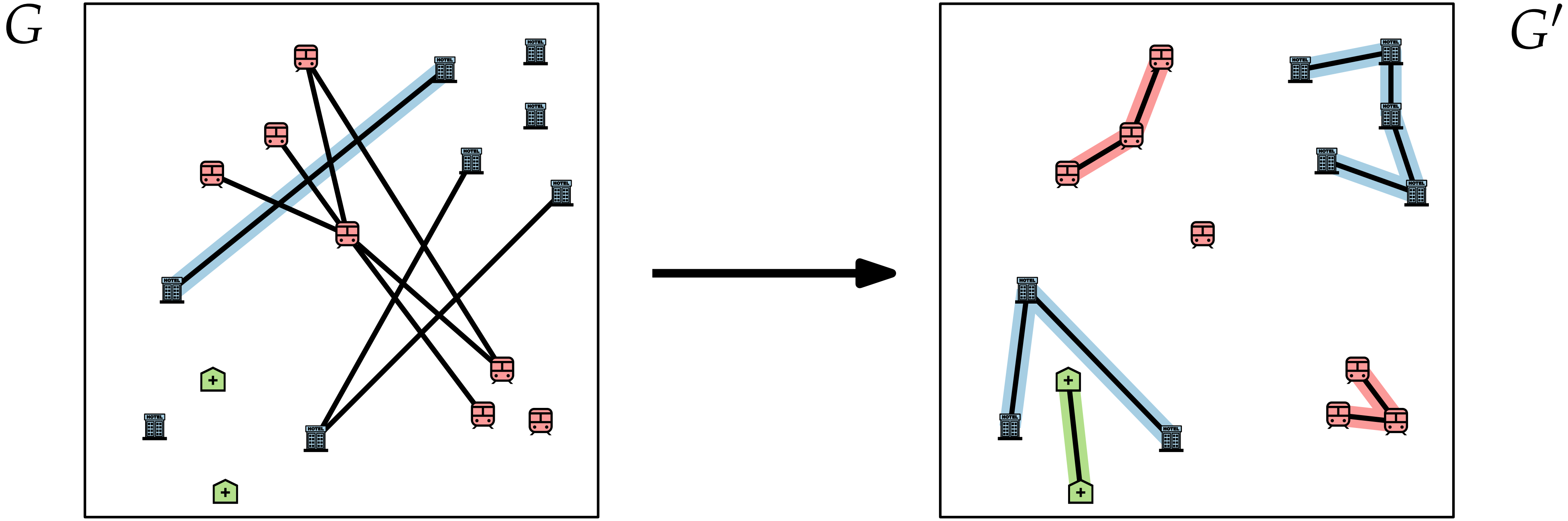


G'

Planar Spanning Forest: Heuristics

ReverseGreedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges

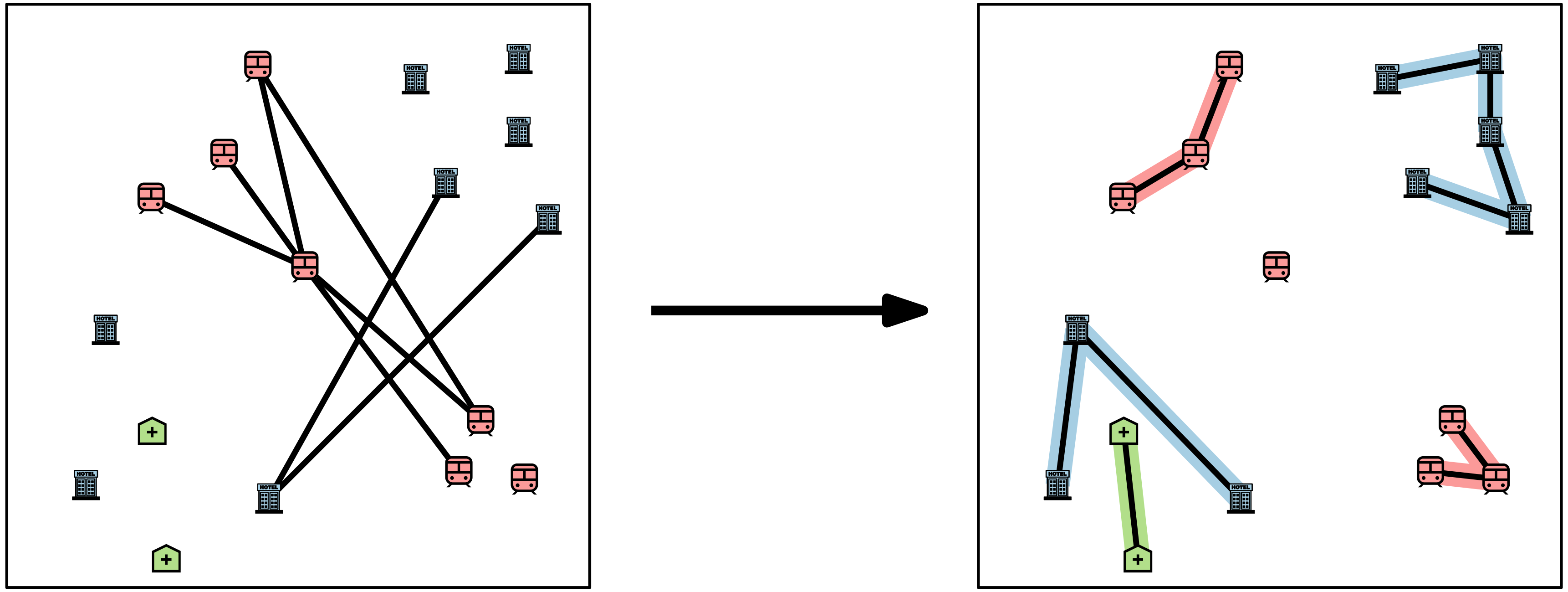


Planar Spanning Forest: Heuristics

ReverseGreedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges

G



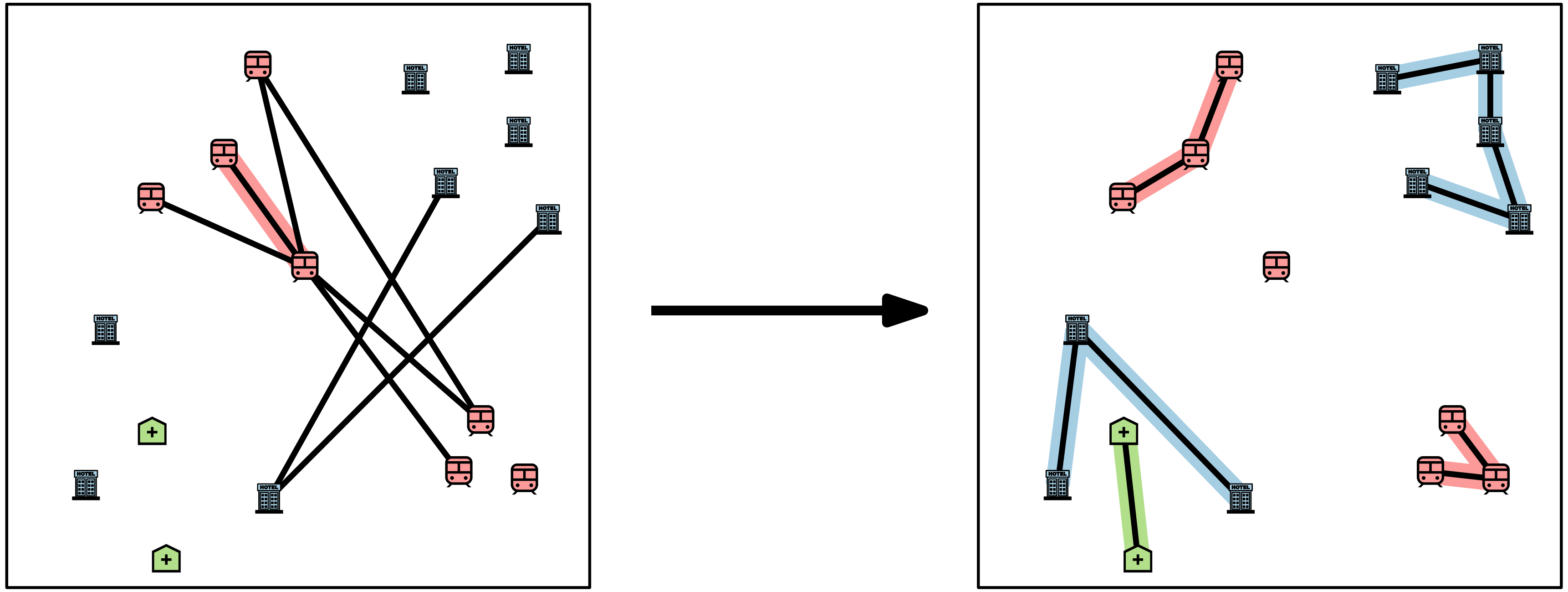
G'

Planar Spanning Forest: Heuristics

ReverseGreedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges

G

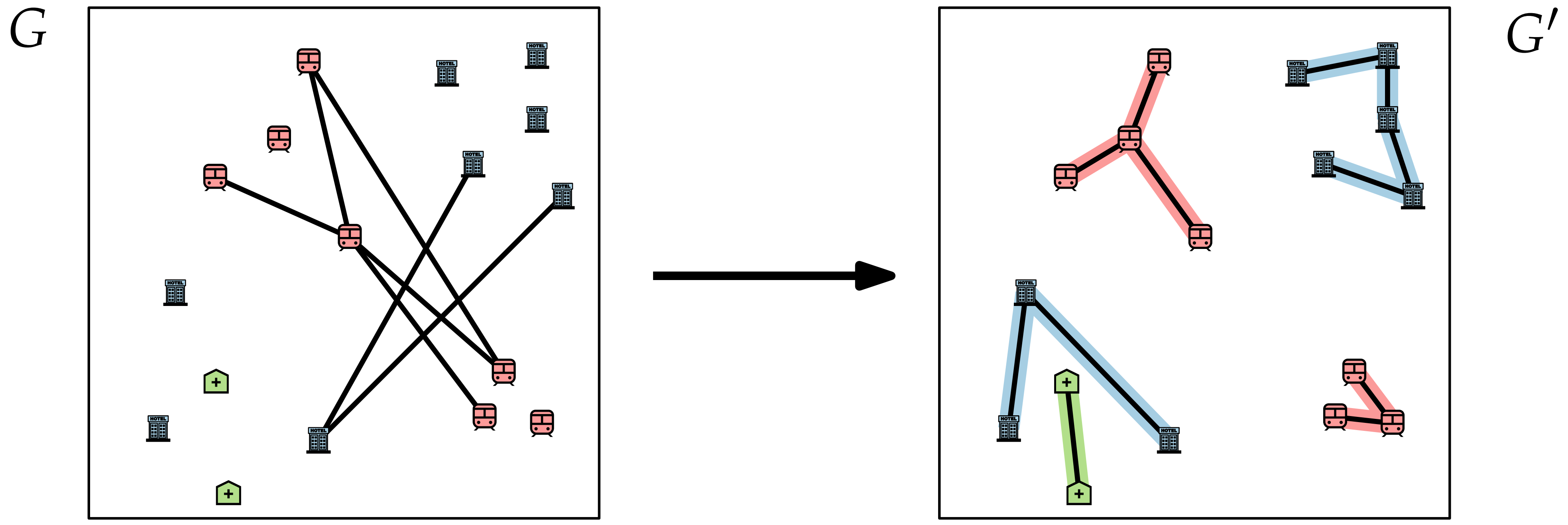


G'

Planar Spanning Forest: Heuristics

REVERSEGREEDY:

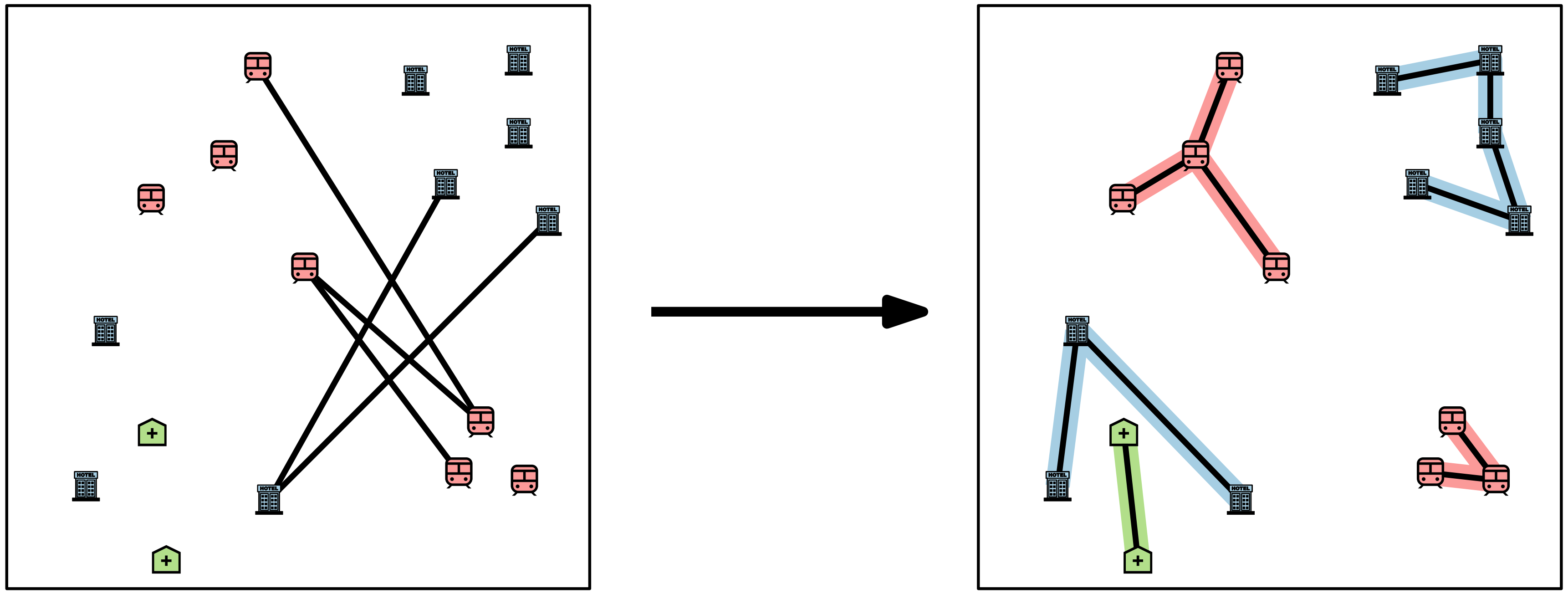
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges

G

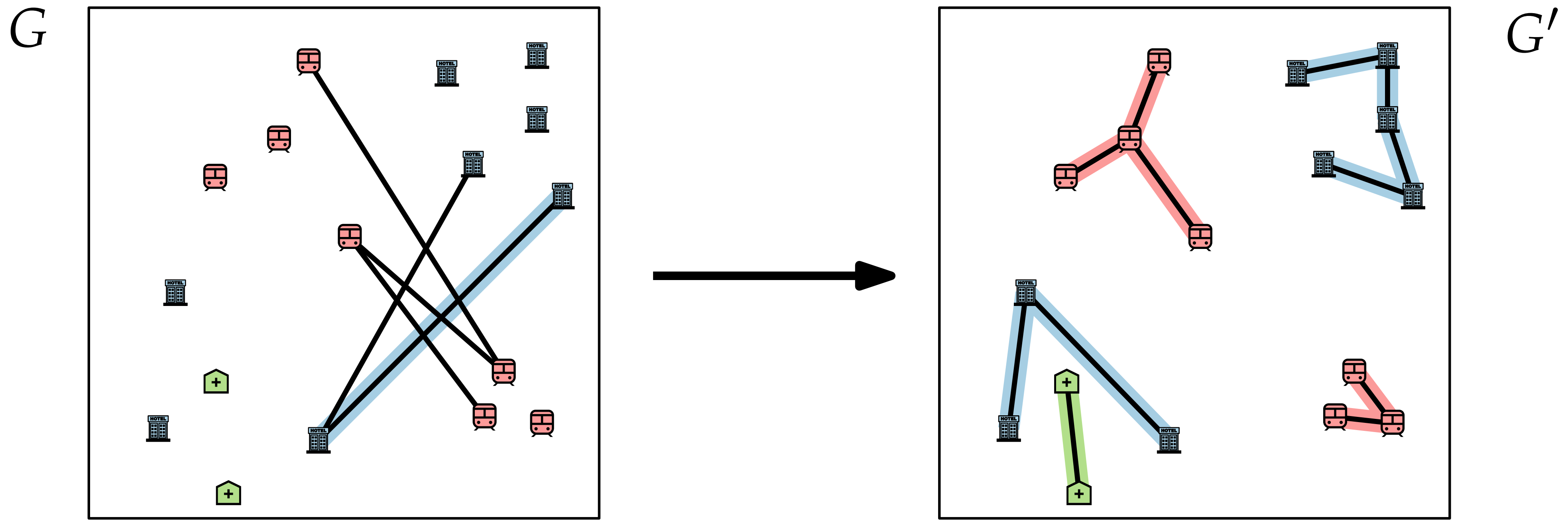


G'

Planar Spanning Forest: Heuristics

ReverseGreedy:

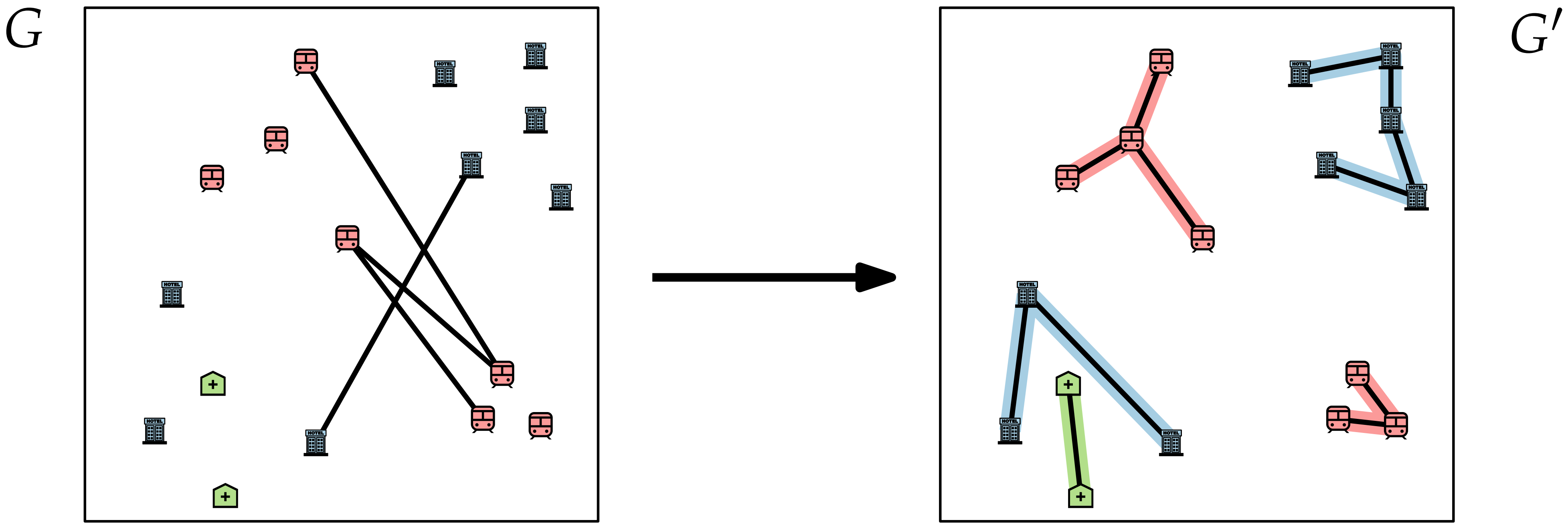
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

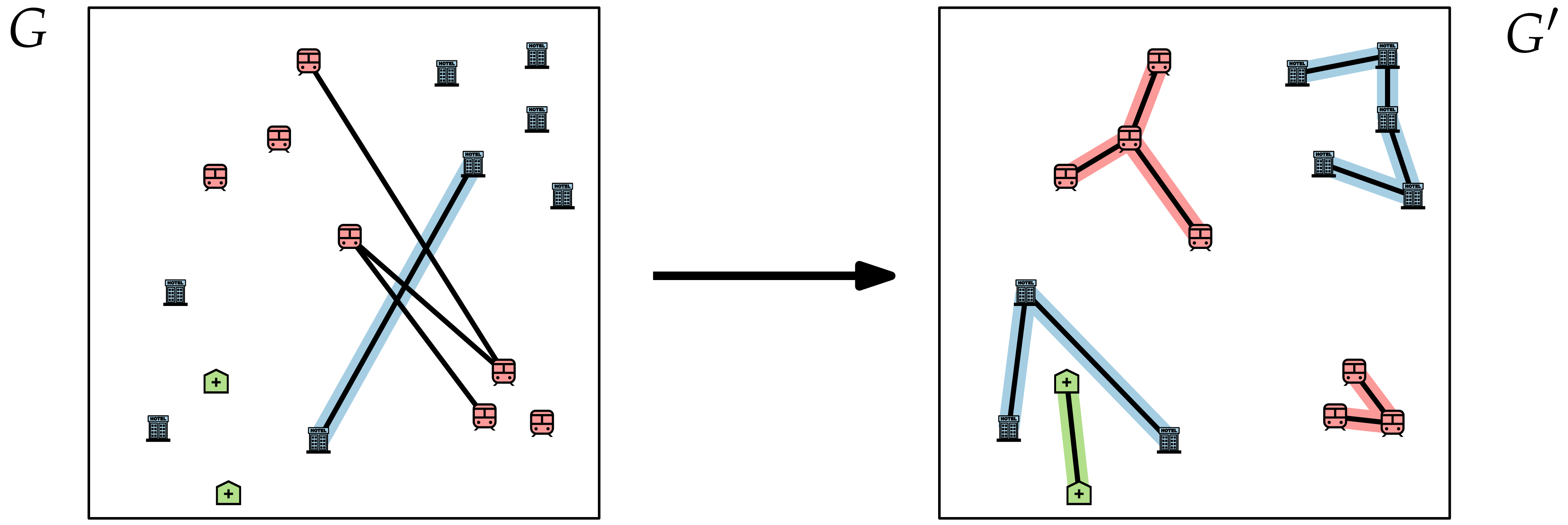
ReverseGreedy:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

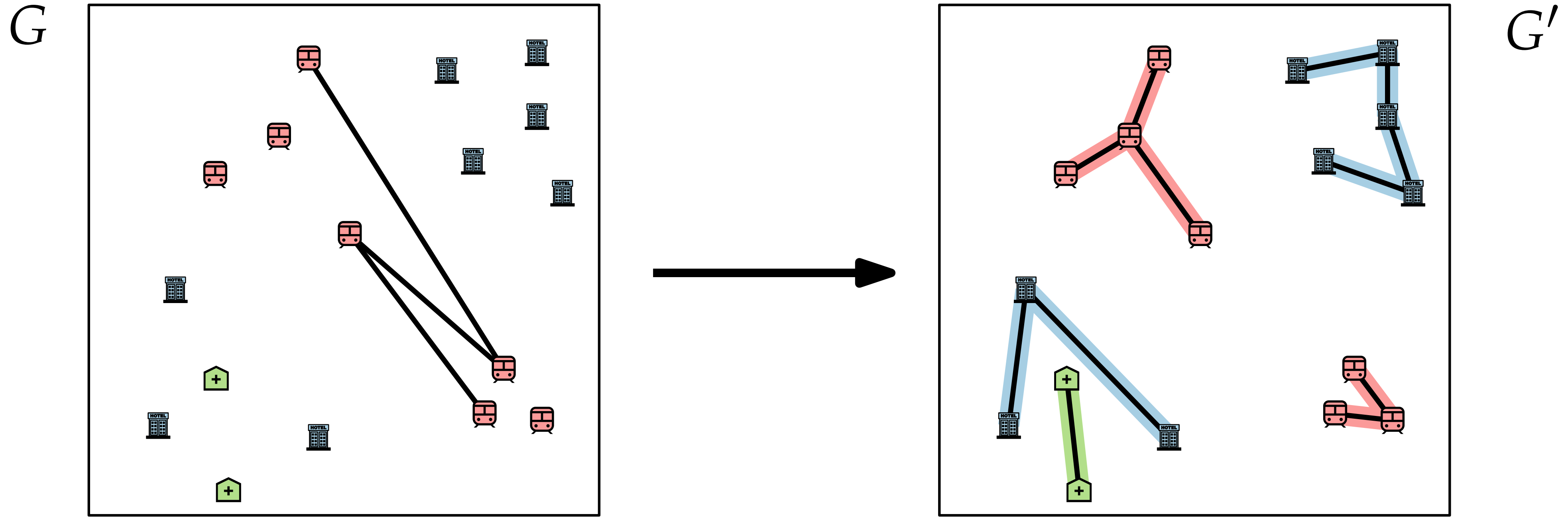
ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

ReverseGreedy:

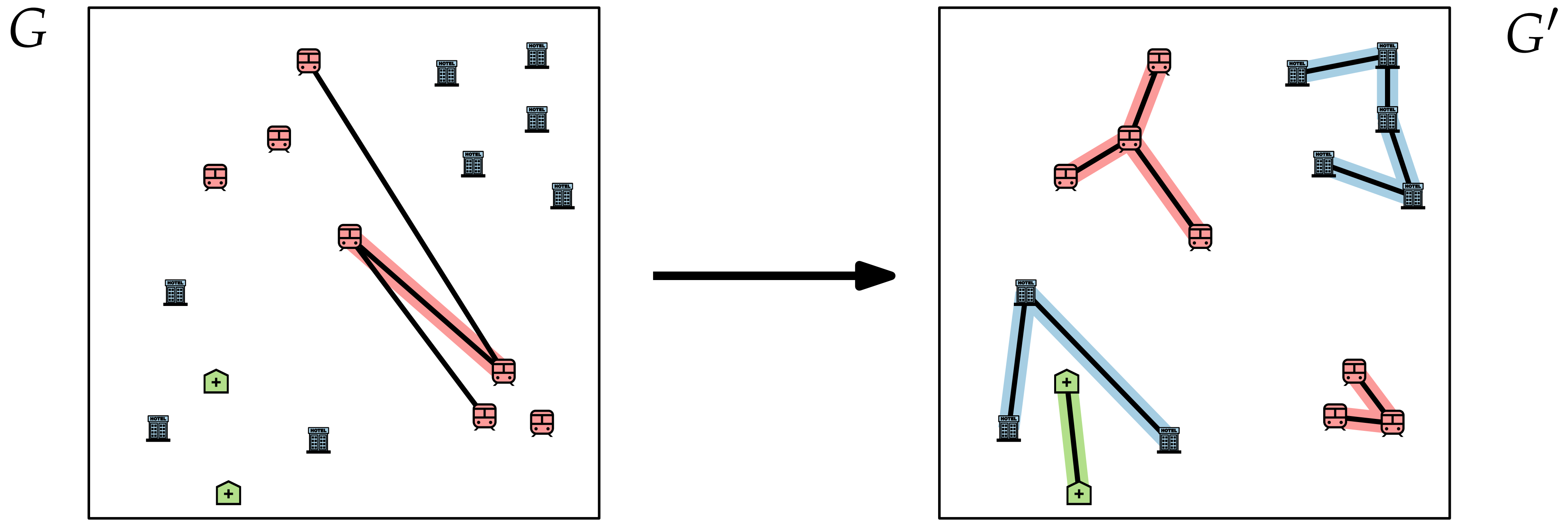
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

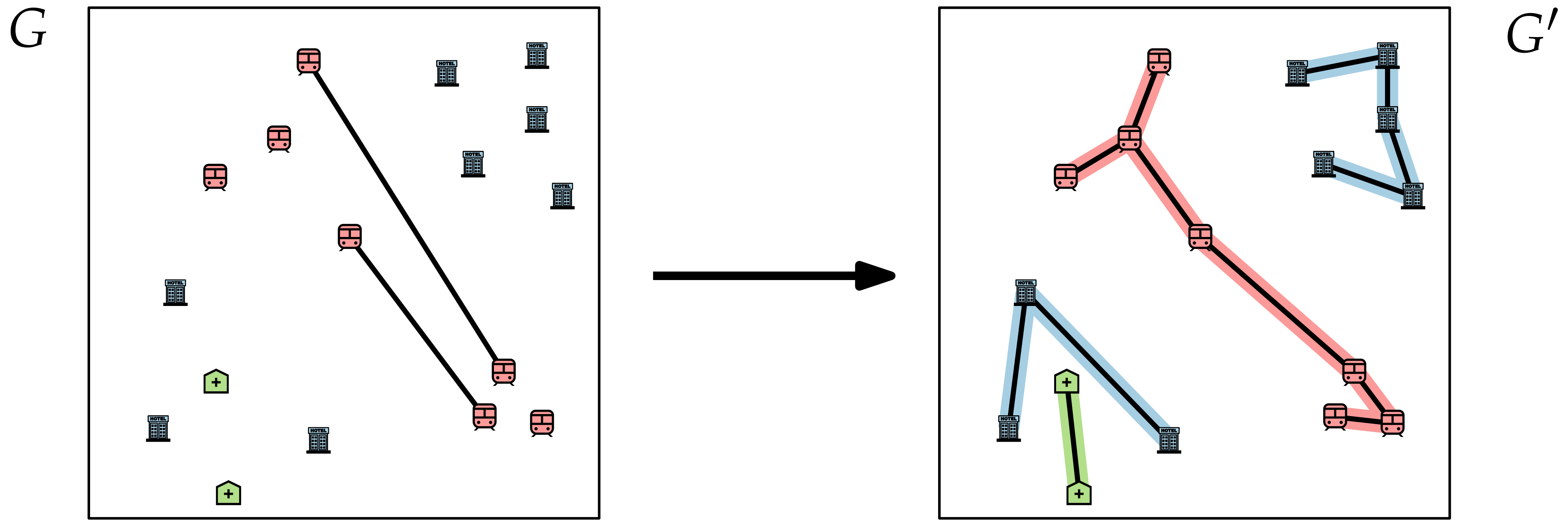
REVERSEGREEDY:

1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

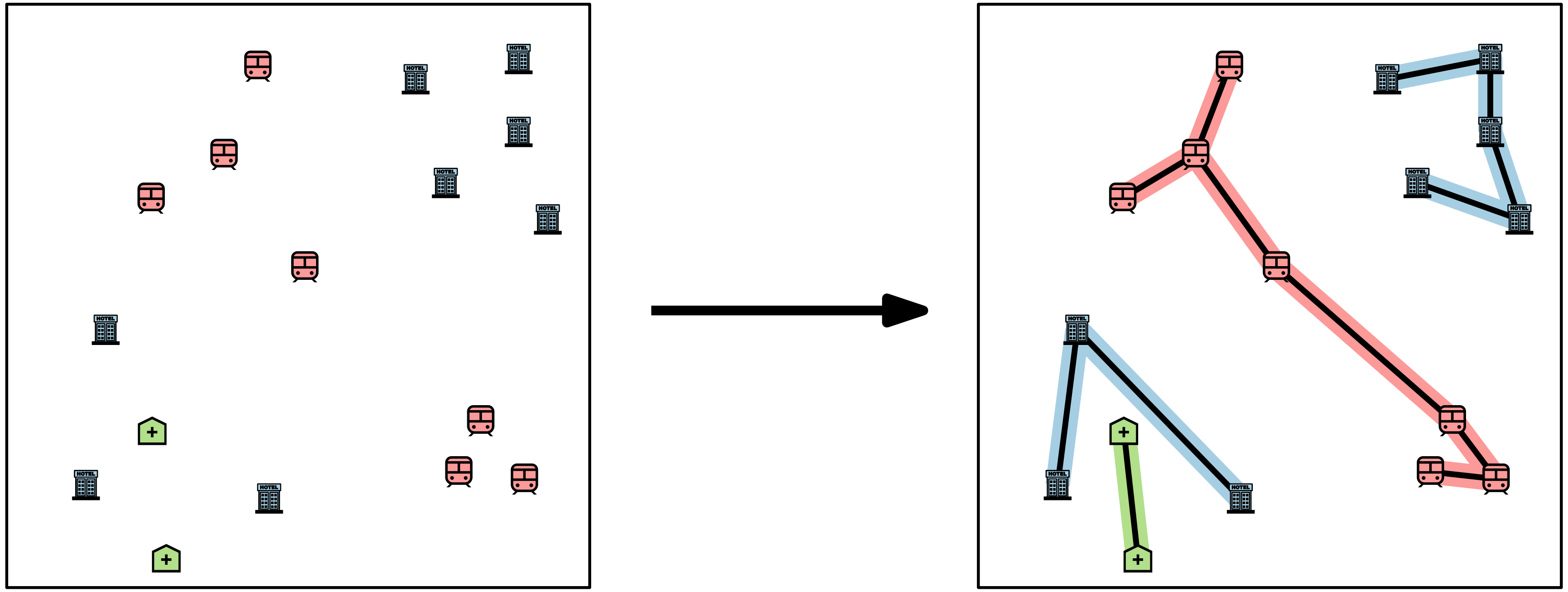
ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges



Planar Spanning Forest: Heuristics

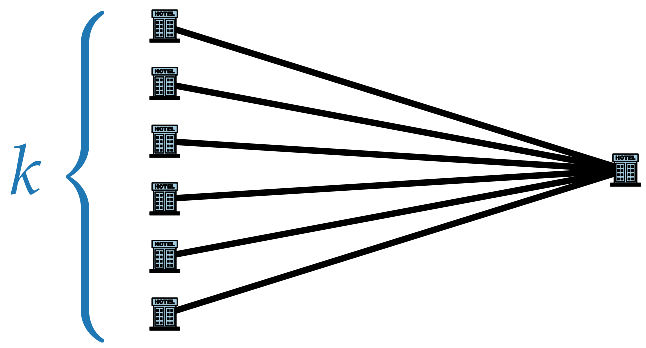
ReverseGreedy:
1. Remove from G every edge that lies within a connected component of G'.
2. Move all non-crossed edges to G' (unless they close a cycle).
3. Remove an edge that is crossed by max. number of edges

G

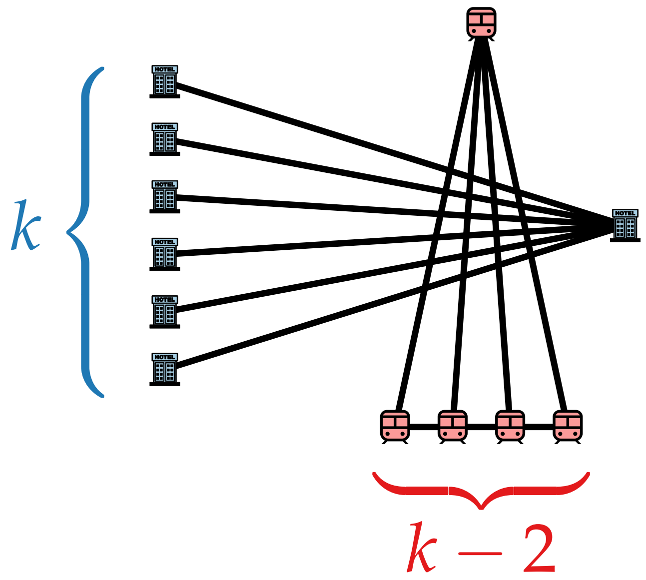


G'

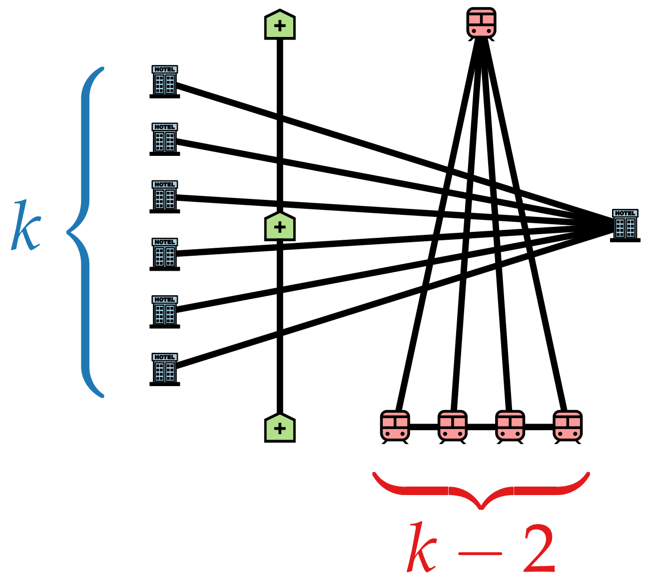
Bad Examples



Bad Examples

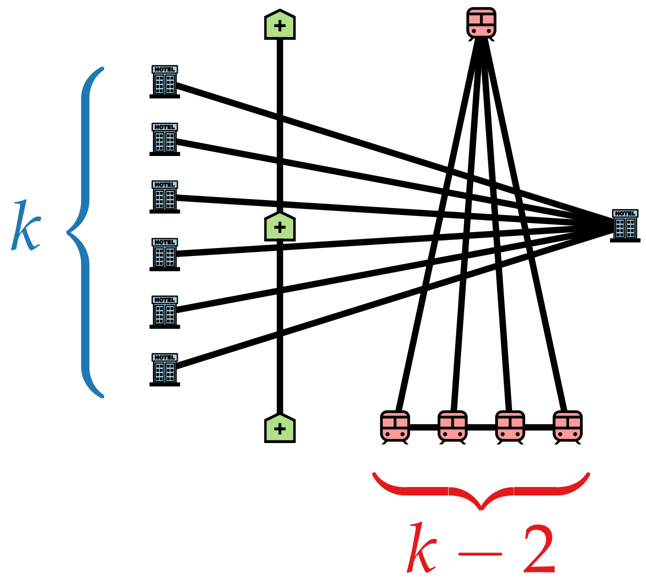


Bad Examples



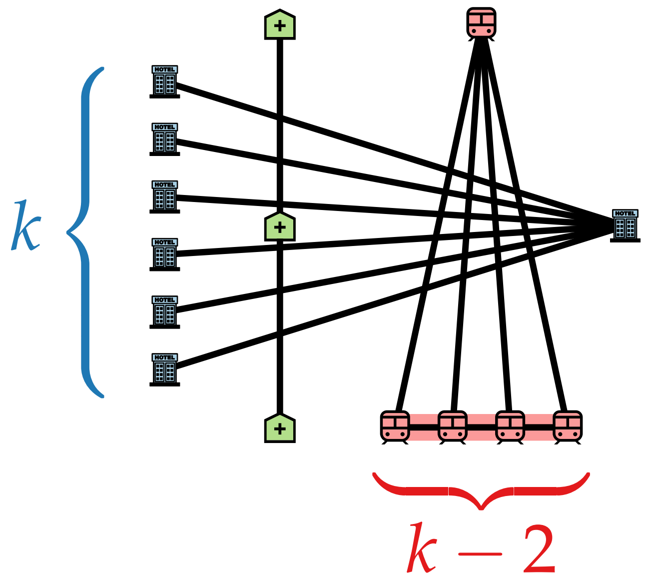
Bad Examples

GREEDY:



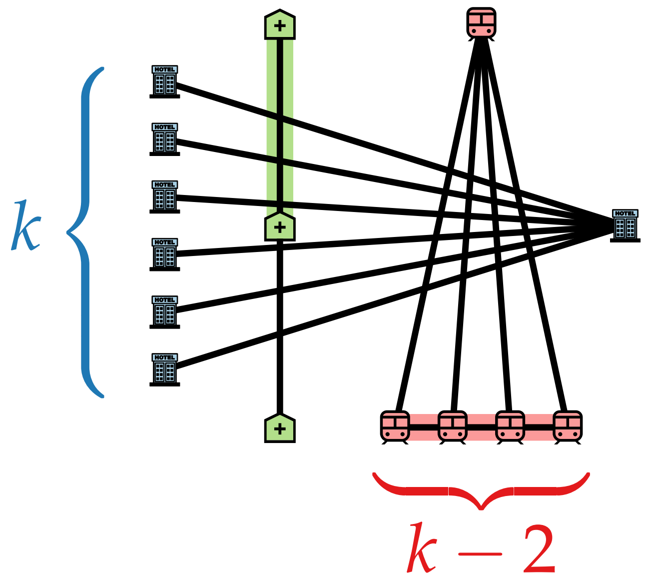
Bad Examples

Greedy:



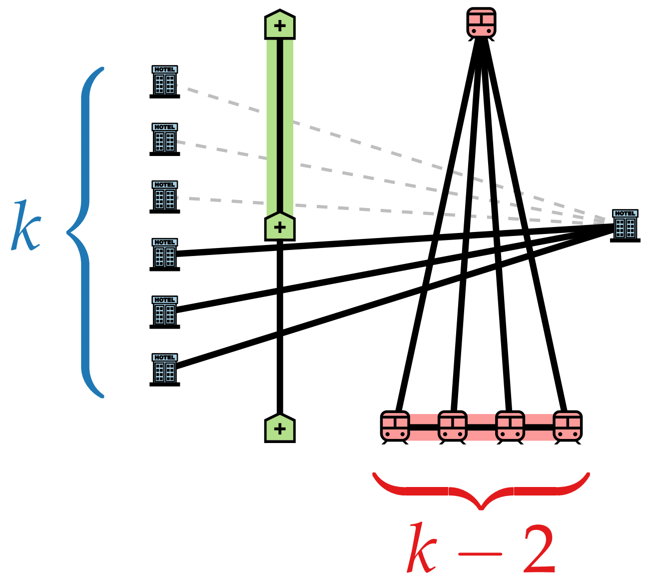
Bad Examples

GREEDY:



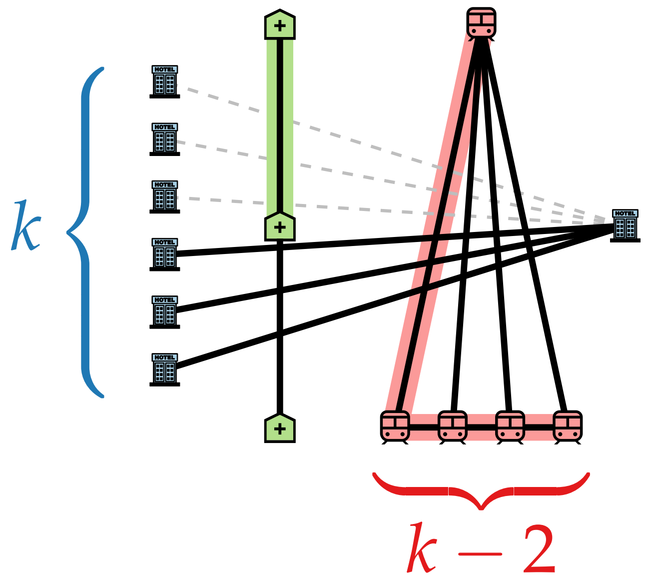
Bad Examples

GREEDY:



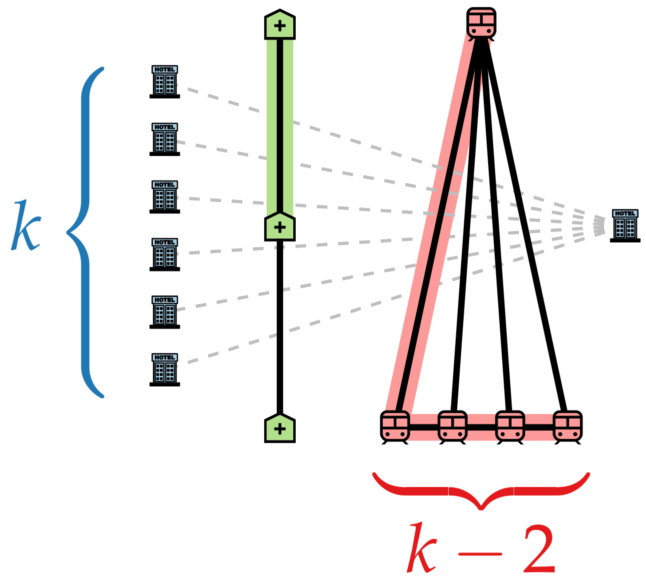
Bad Examples

Greedy:



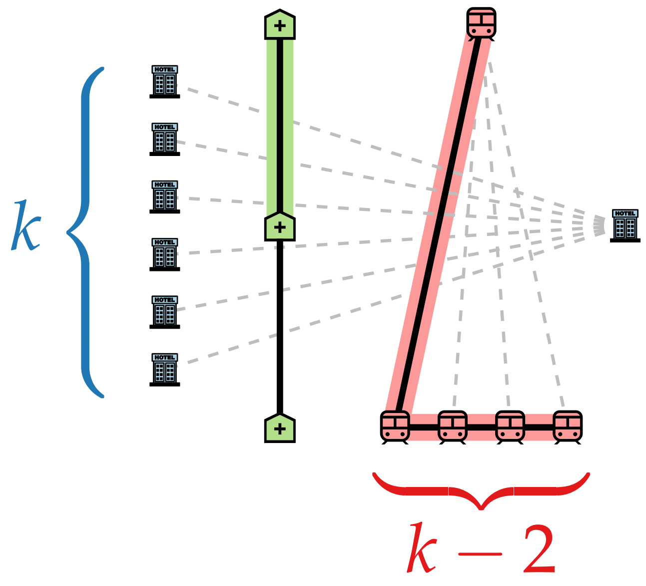
Bad Examples

GREEDY:



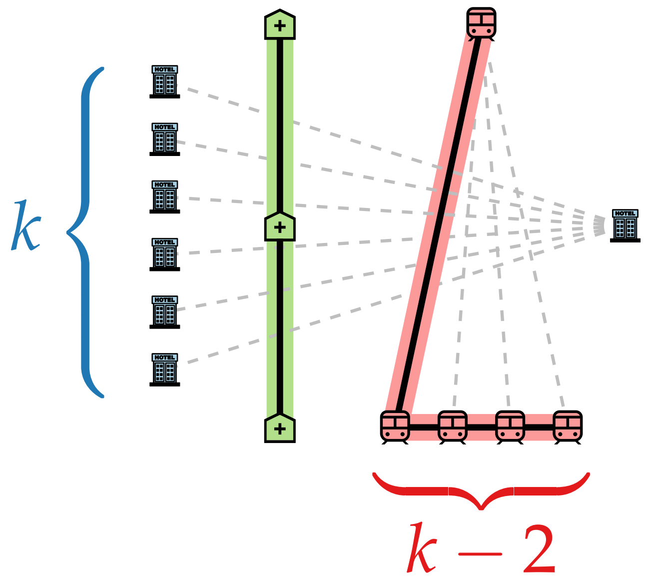
Bad Examples

GREEDY:



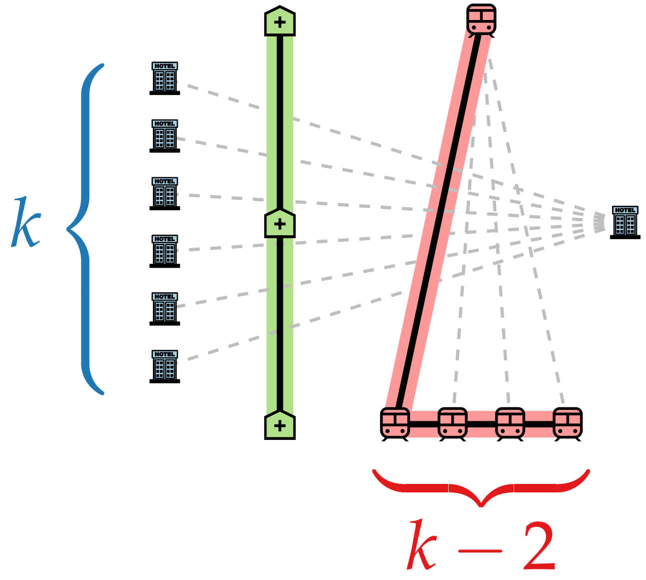
Bad Examples

Greedy:

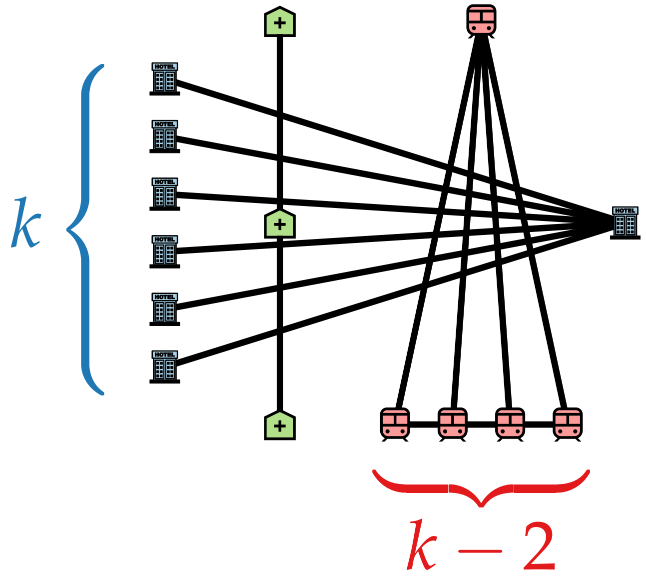


Bad Examples

Greedy:

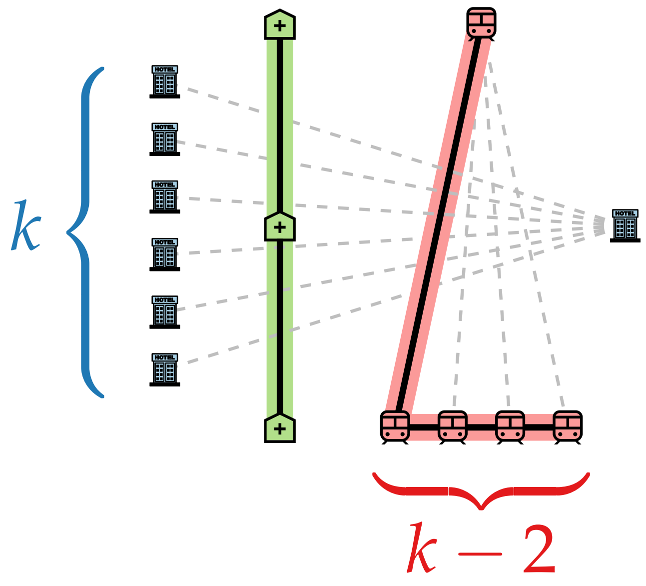


ReverseGreedy:

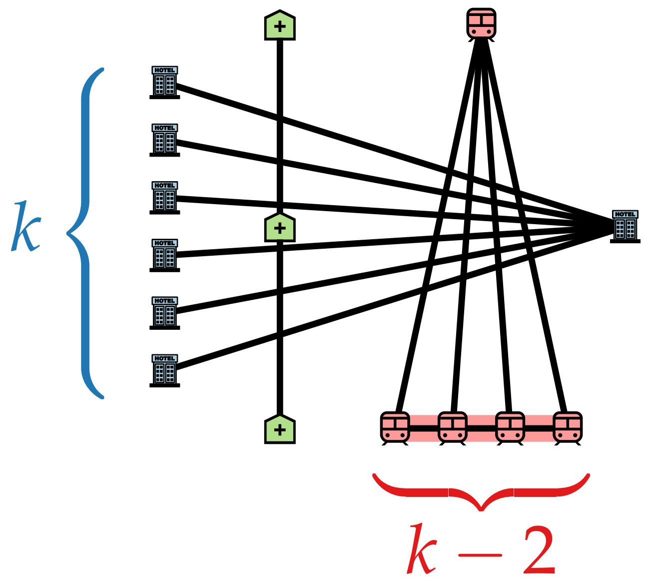


Bad Examples

Greedy:

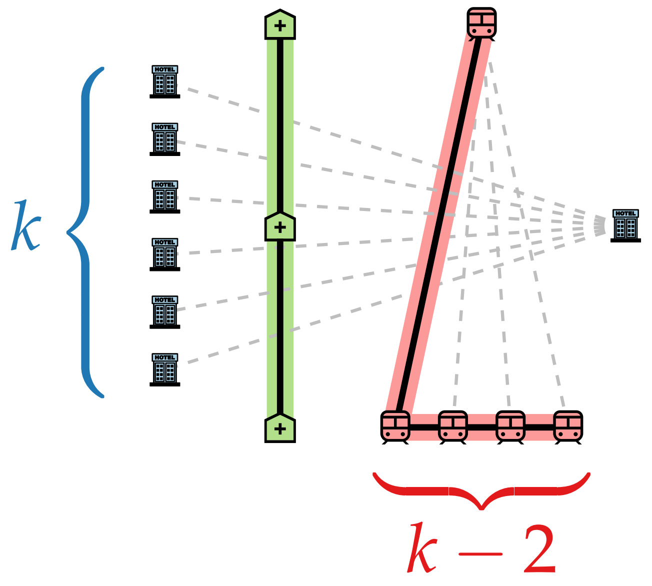


ReverseGreedy:

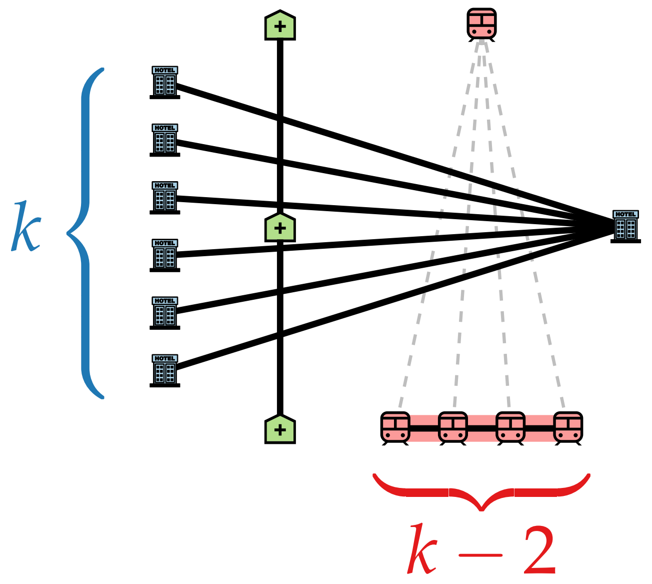


Bad Examples

Greedy:

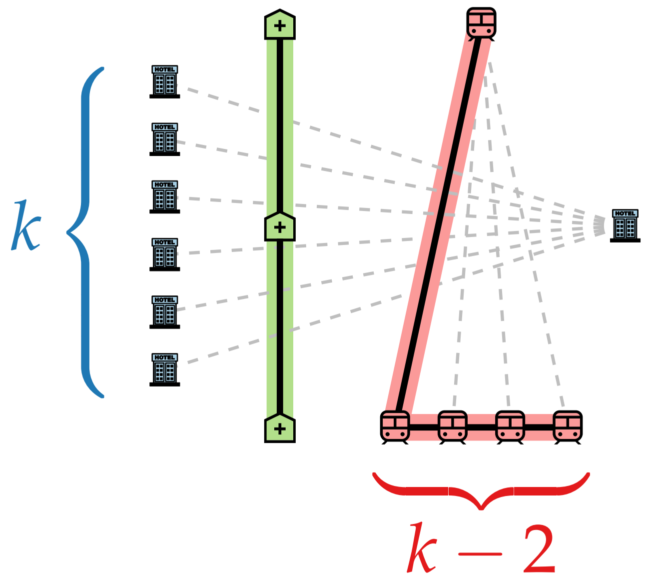


ReverseGreedy:

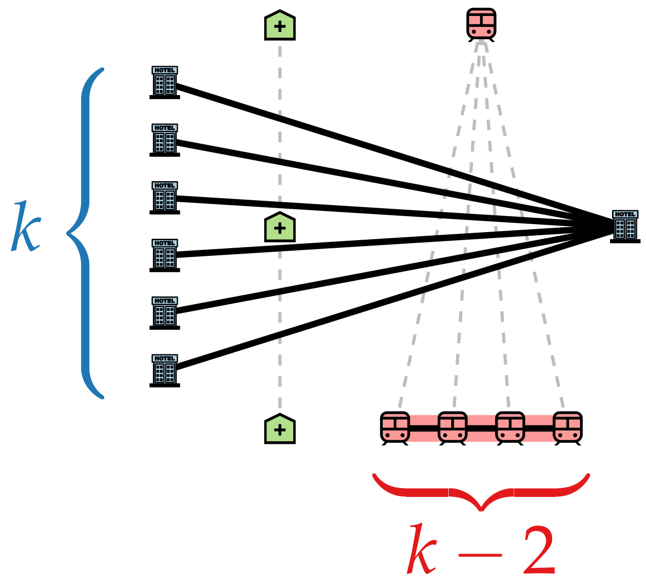


Bad Examples

GREEDY:

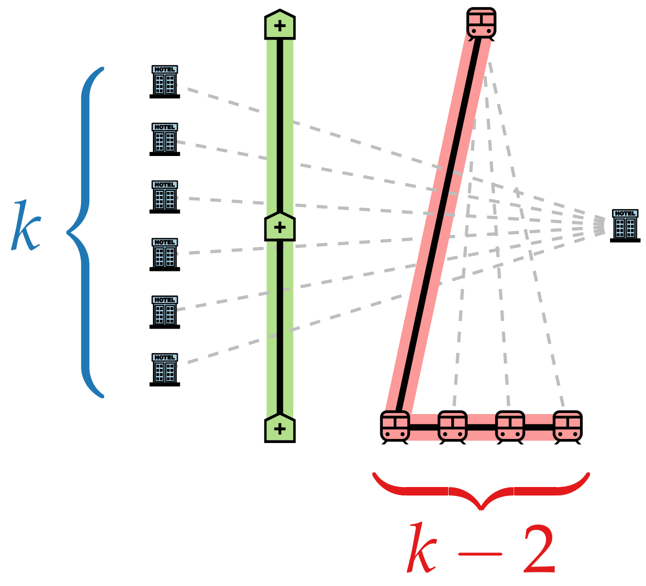


REVERSEGREEDY:

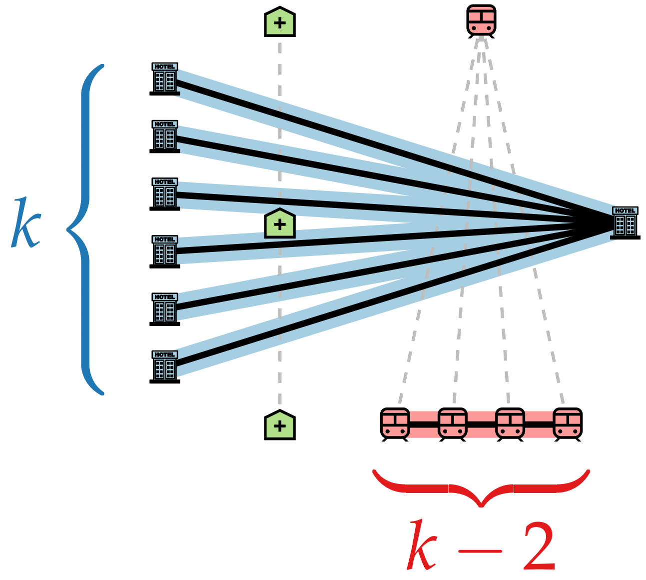


Bad Examples

Greedy:

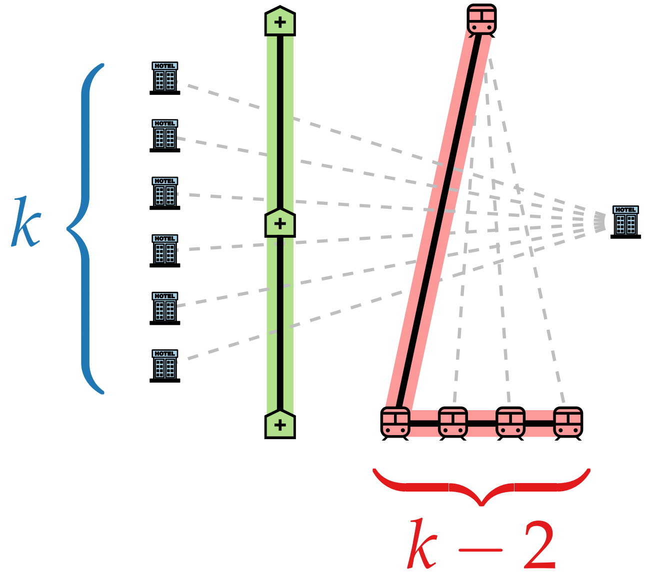


ReverseGreedy:

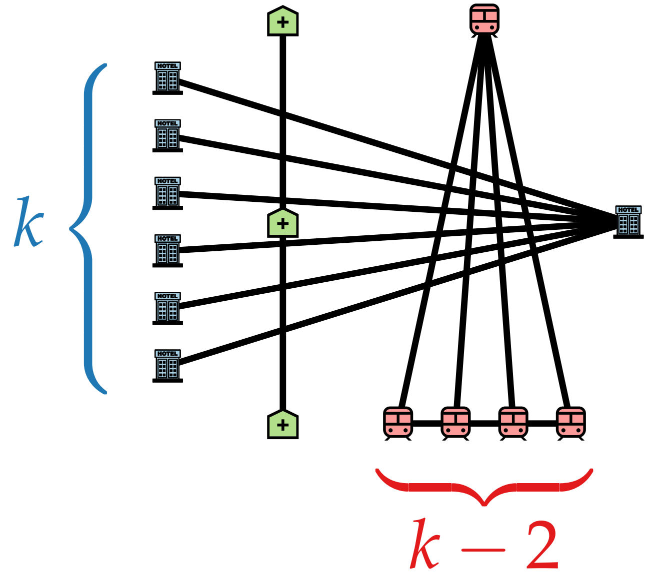


Bad Examples

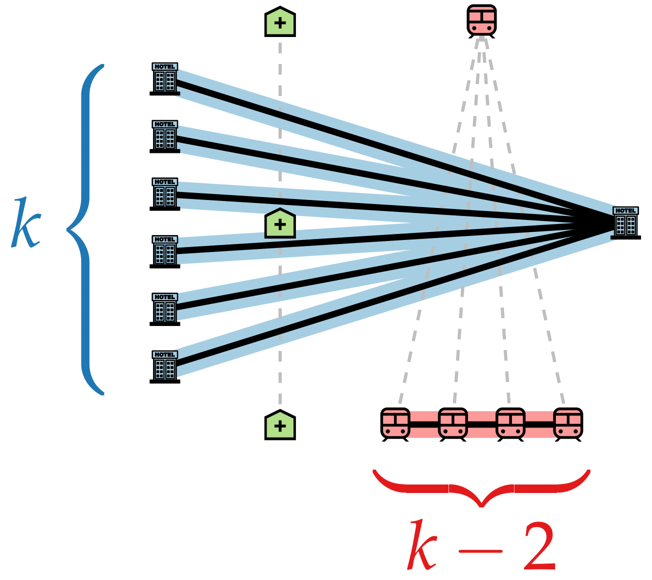
Greedy:



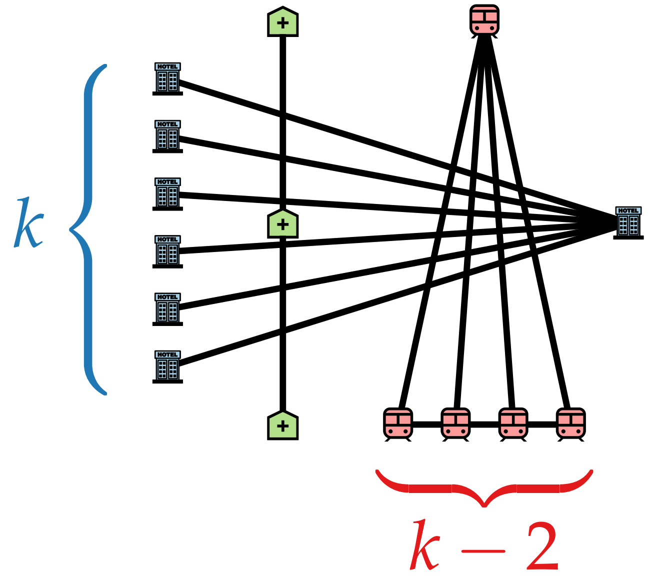
Greedy:



ReverseGreedy:

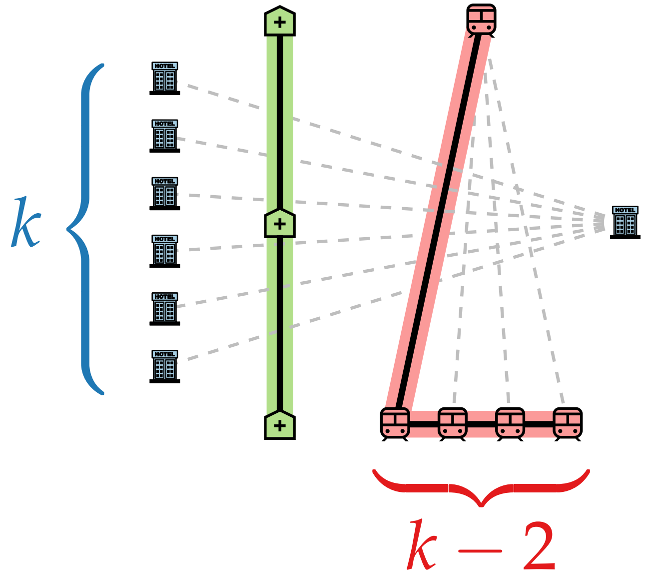


ReverseGreedy:

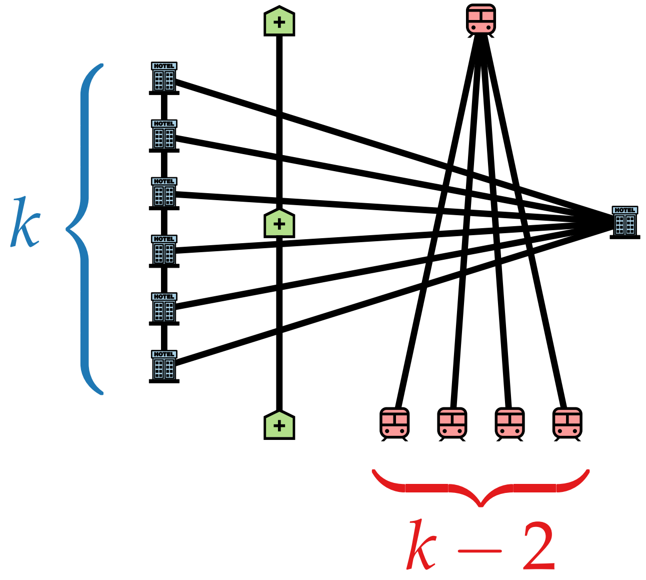


Bad Examples

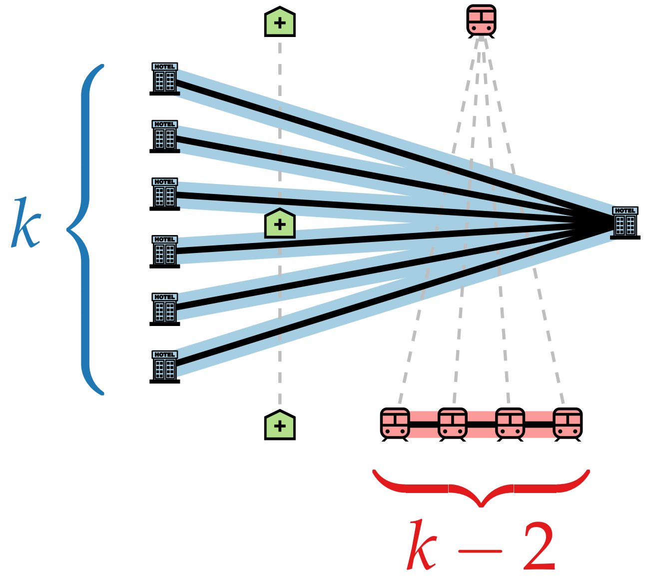
GREEDY:



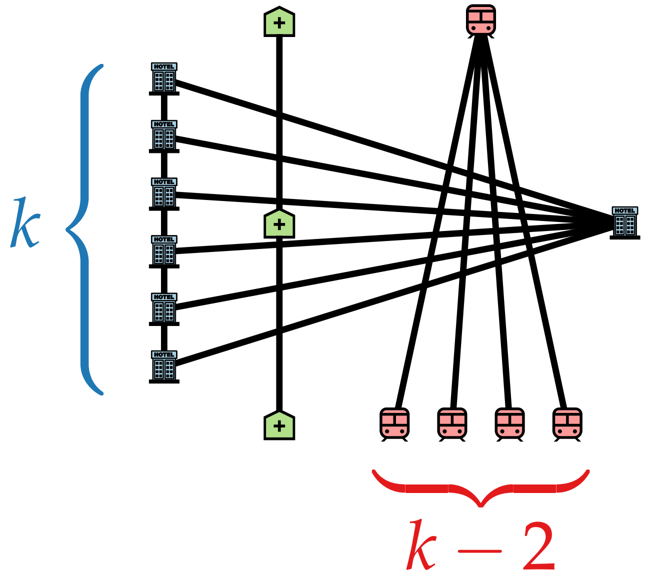
GREEDY:



REVERSEGREEDY:

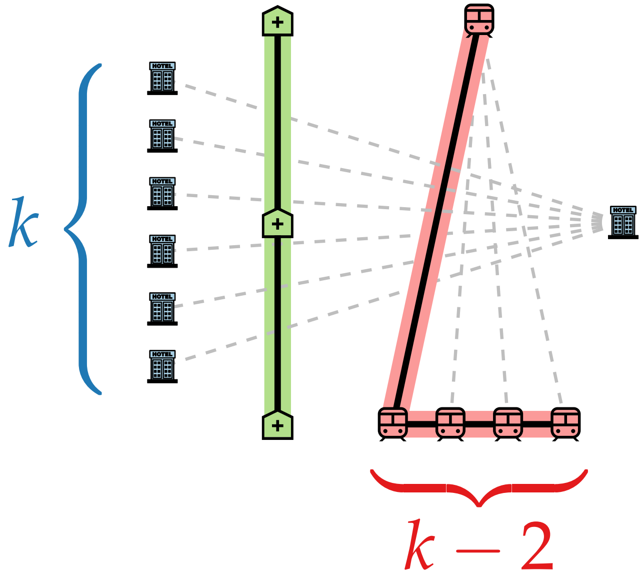


REVERSEGREEDY:

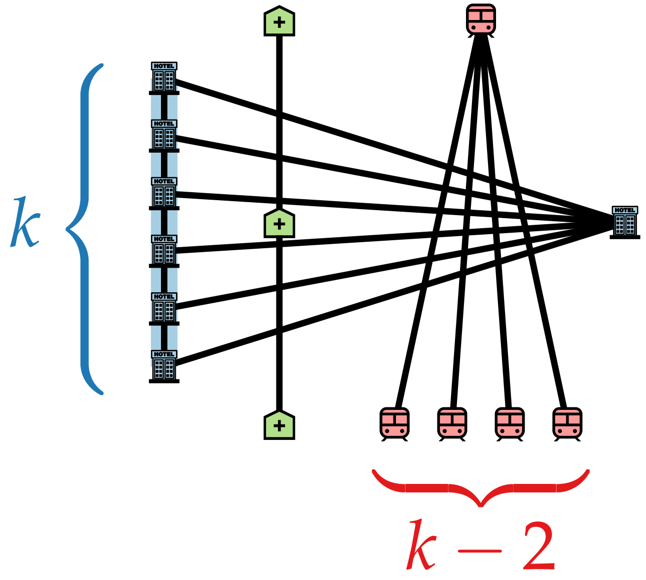


Bad Examples

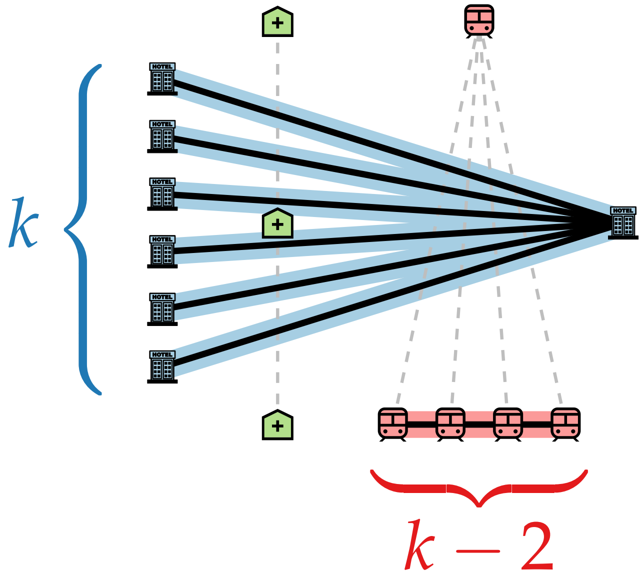
Greedy:



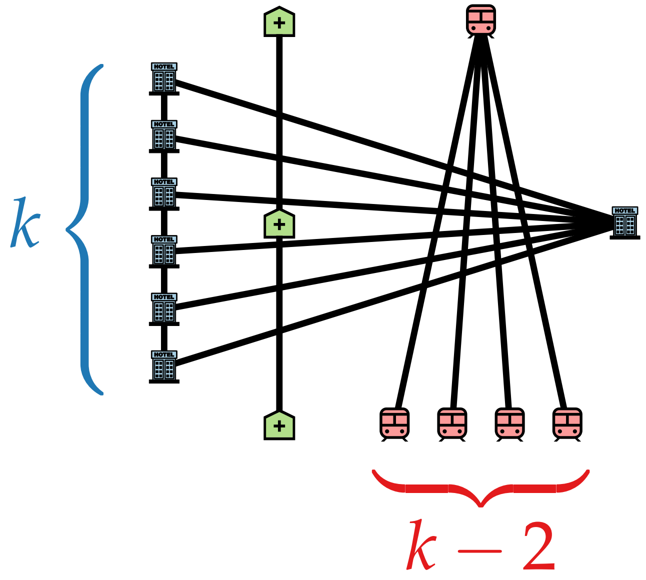
Greedy:



ReverseGreedy:

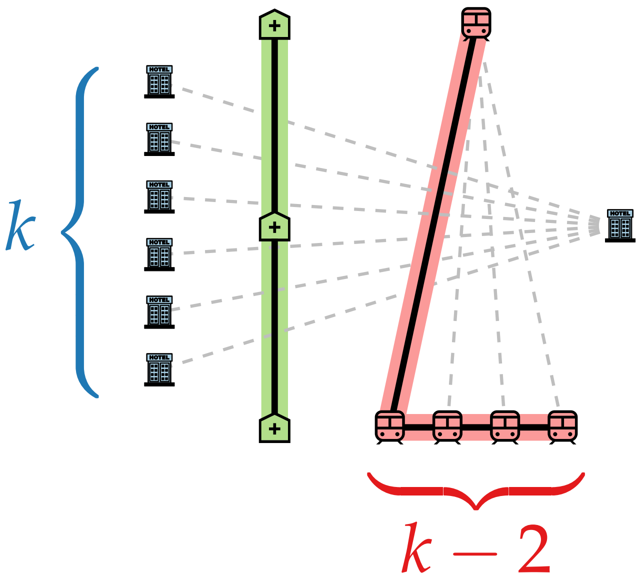


ReverseGreedy:

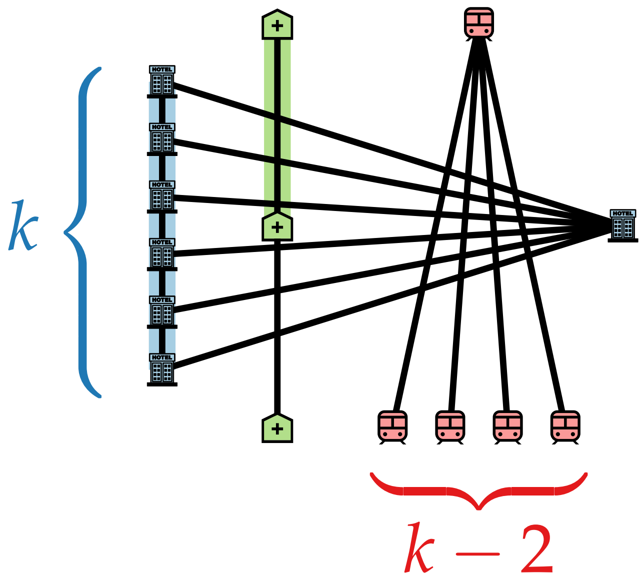


Bad Examples

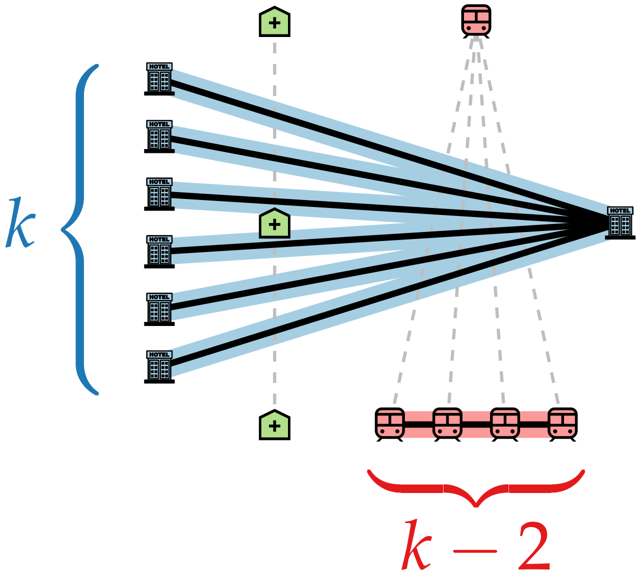
GREEDY:



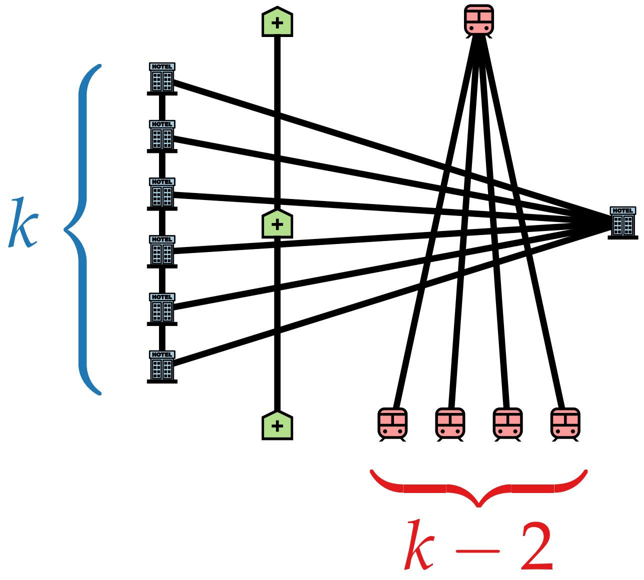
GREEDY:



REVERSEGREEDY:

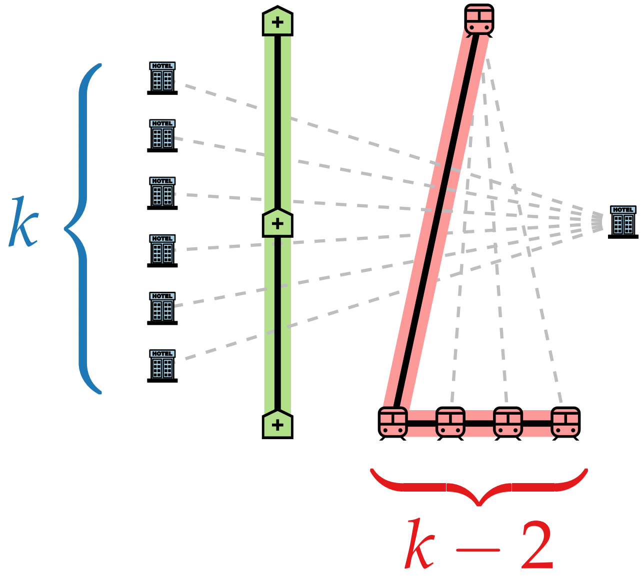


REVERSEGREEDY:

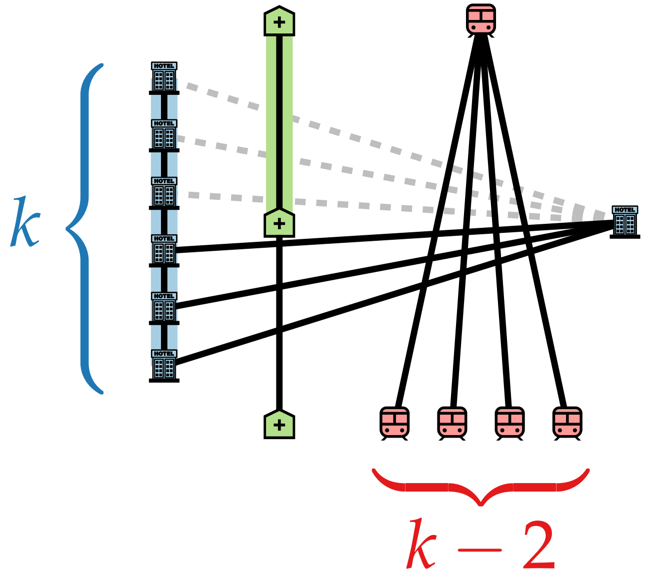


Bad Examples

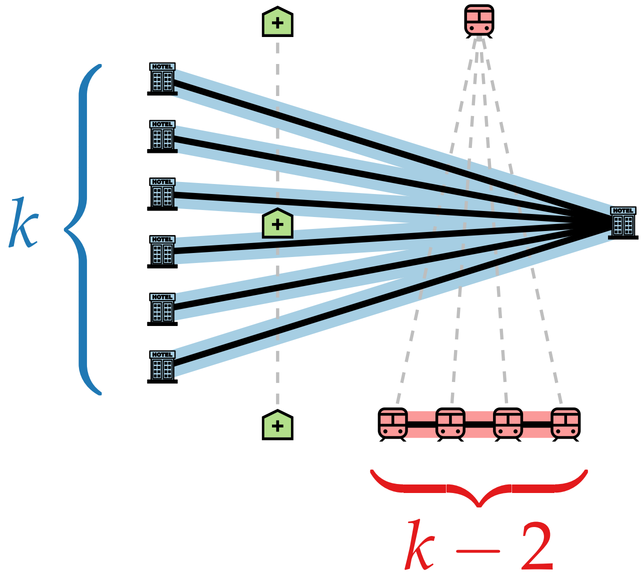
GREEDY:



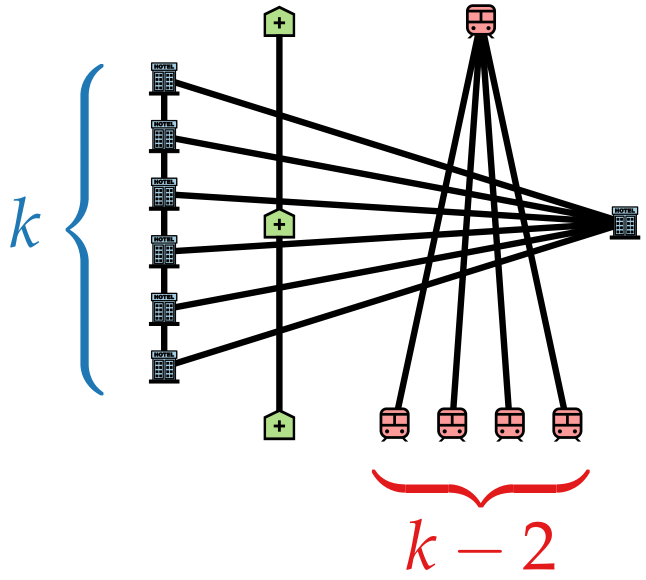
GREEDY:



REVERSEGREEDY:

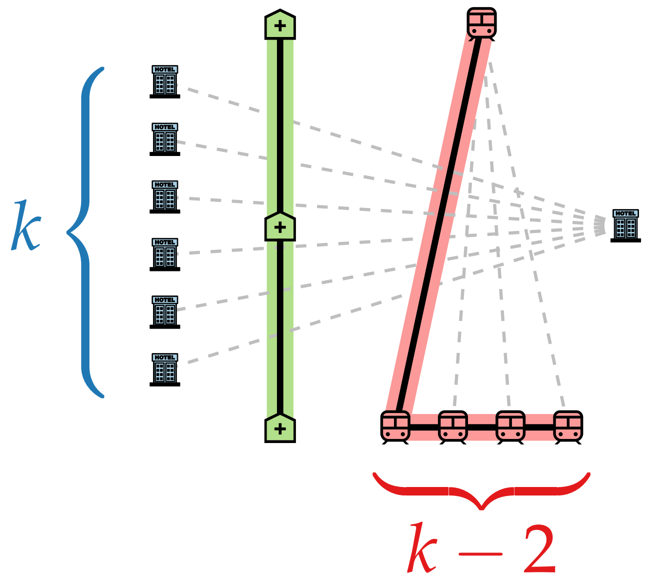


REVERSEGREEDY:

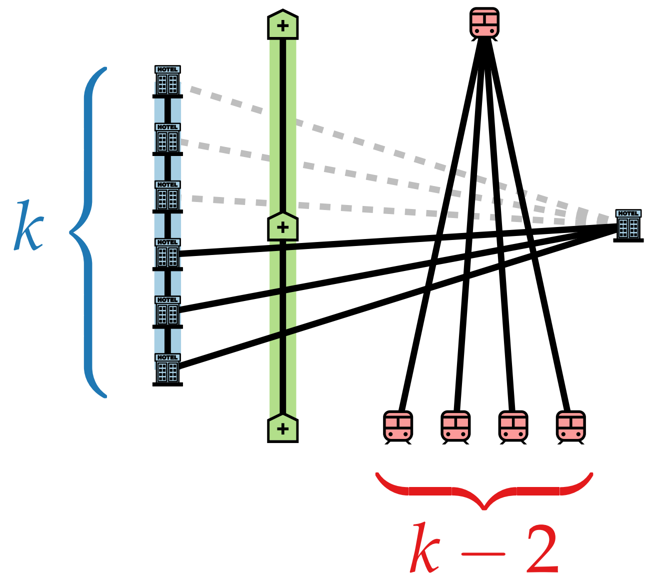


Bad Examples

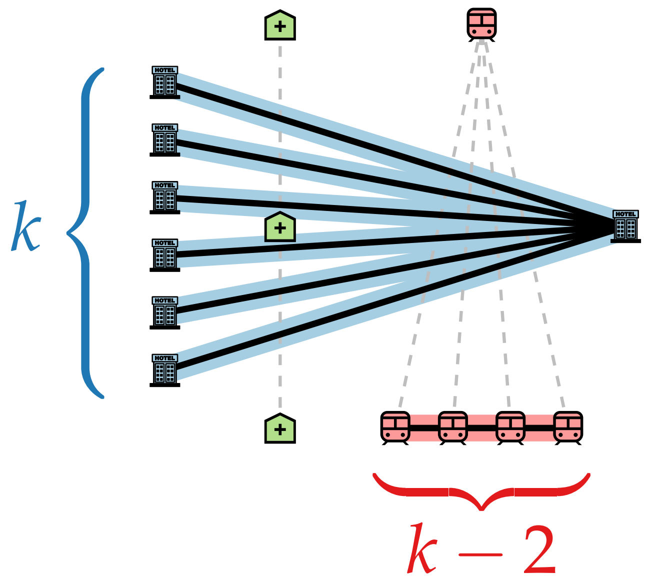
GREEDY:



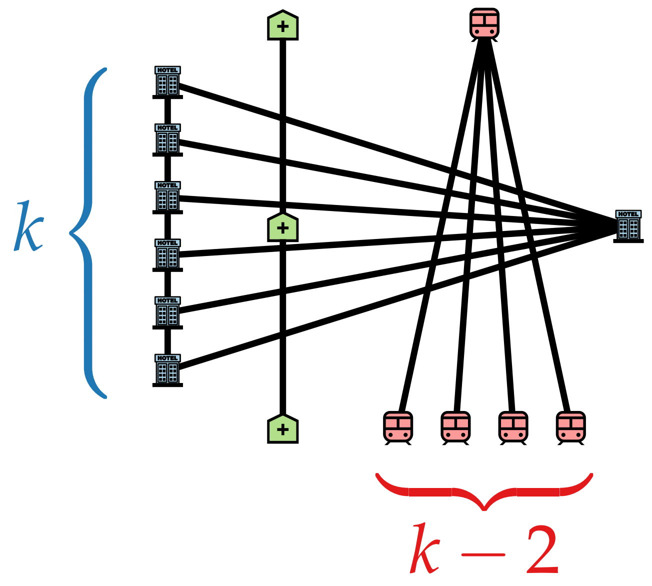
GREEDY:



REVERSEGREEDY:

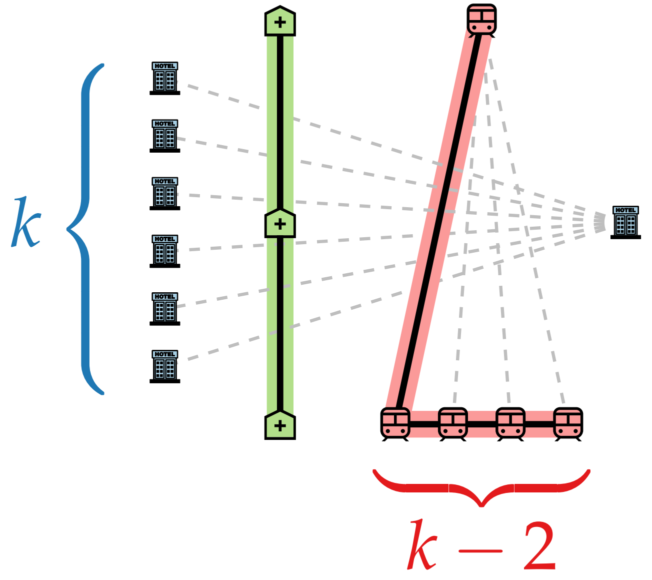


REVERSEGREEDY:

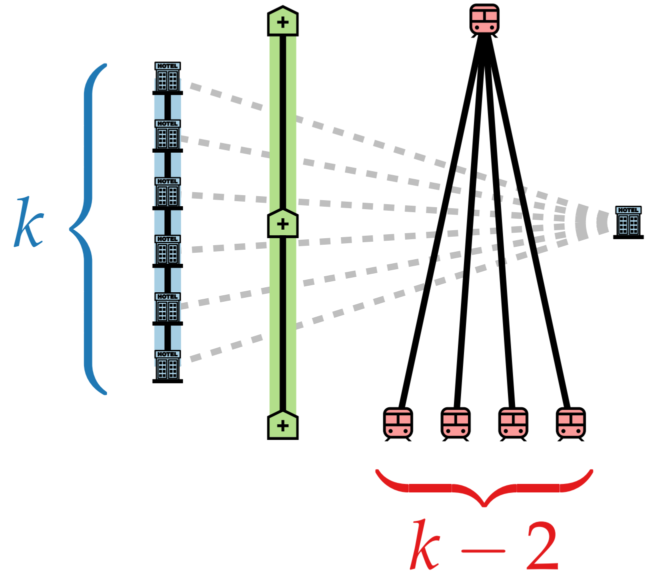


Bad Examples

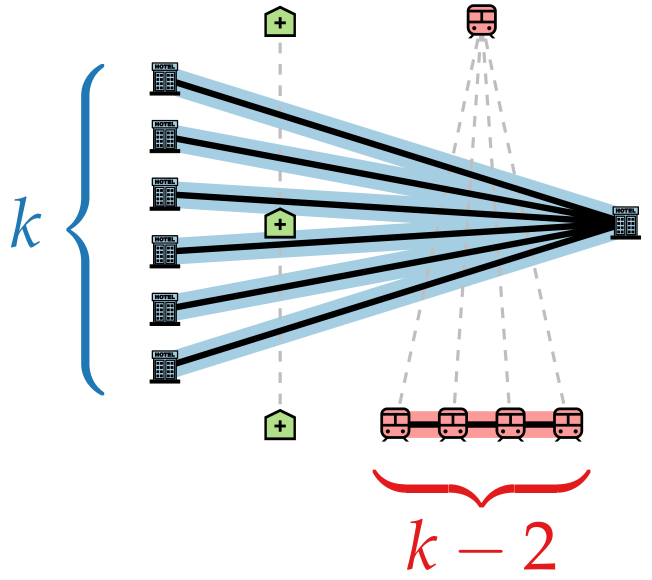
GREEDY:



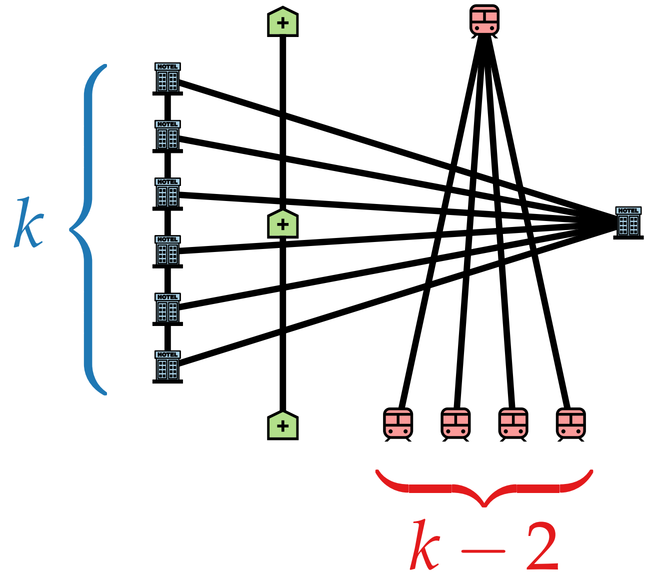
GREEDY:



REVERSEGREEDY:

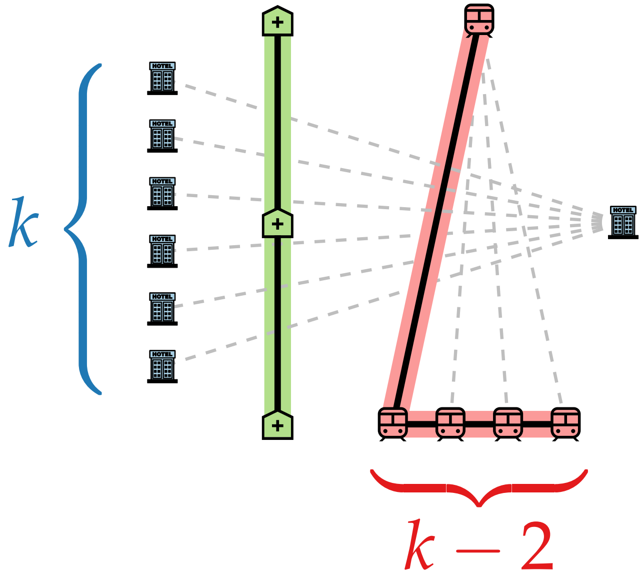


REVERSEGREEDY:

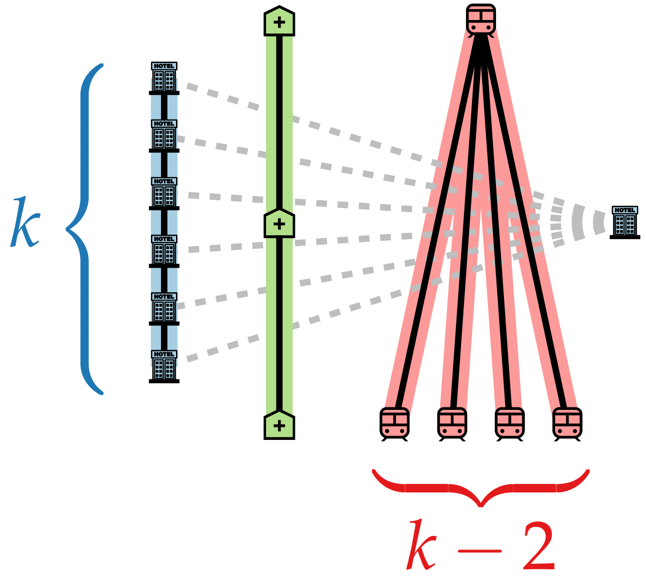


Bad Examples

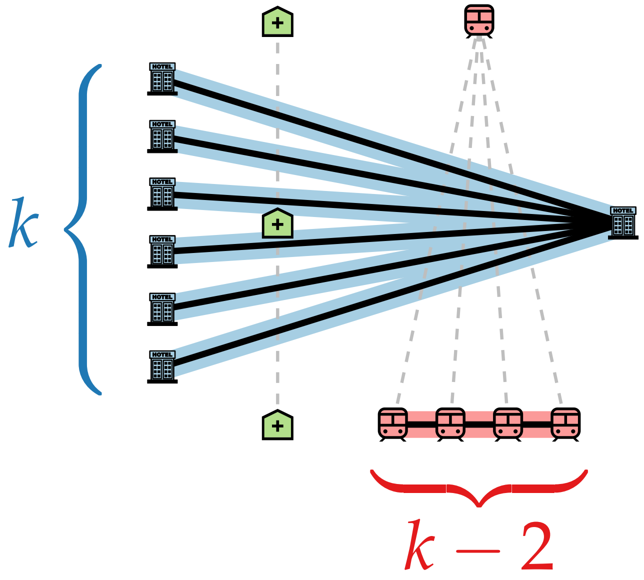
Greedy:



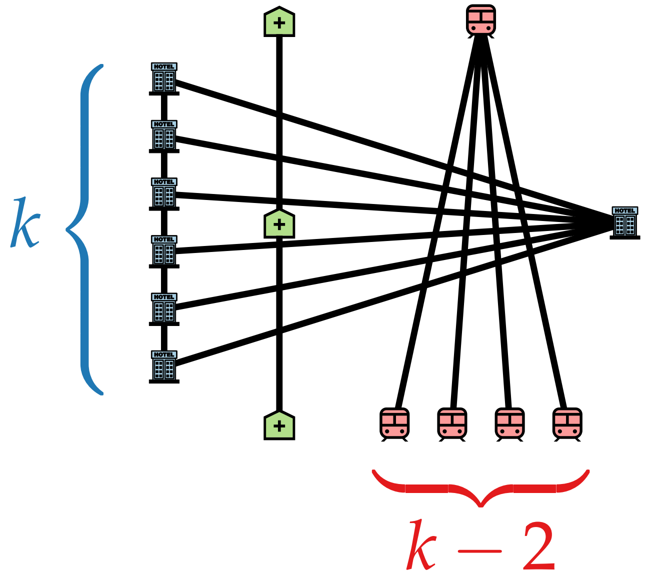
Greedy:



ReverseGreedy:

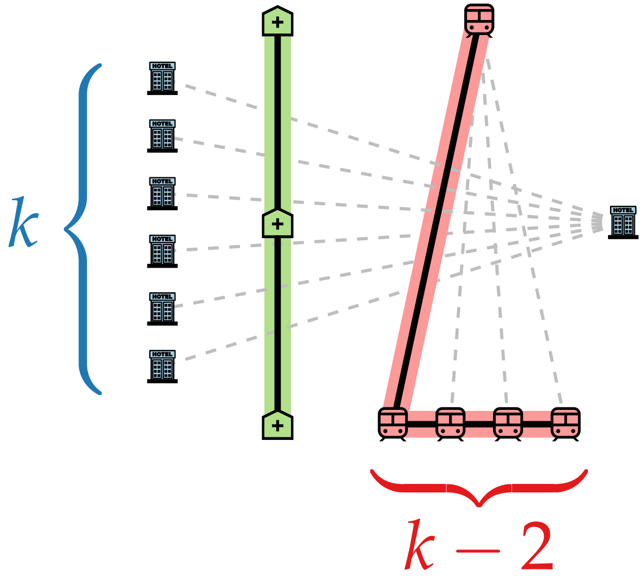


ReverseGreedy:

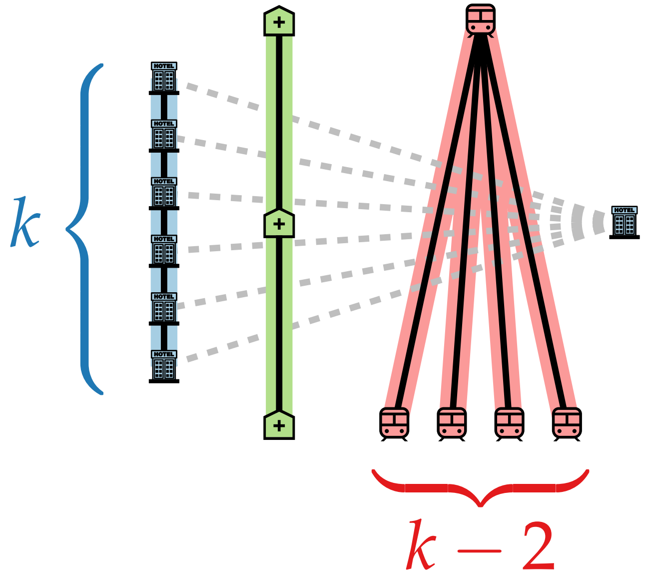


Bad Examples

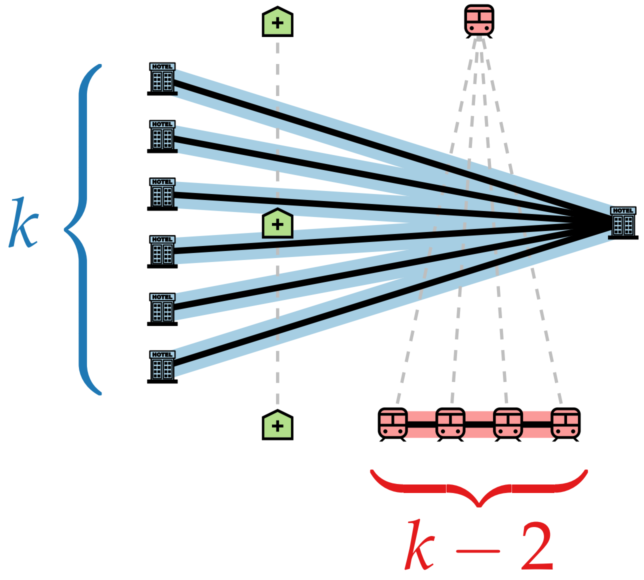
Greedy:



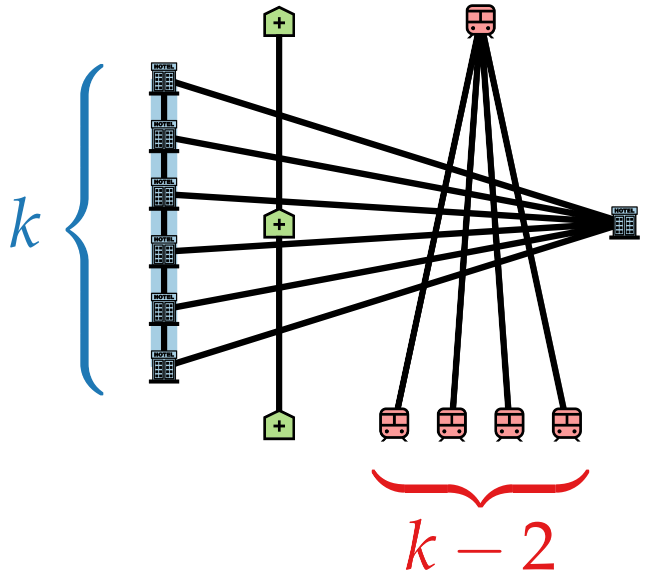
Greedy:



ReverseGreedy:

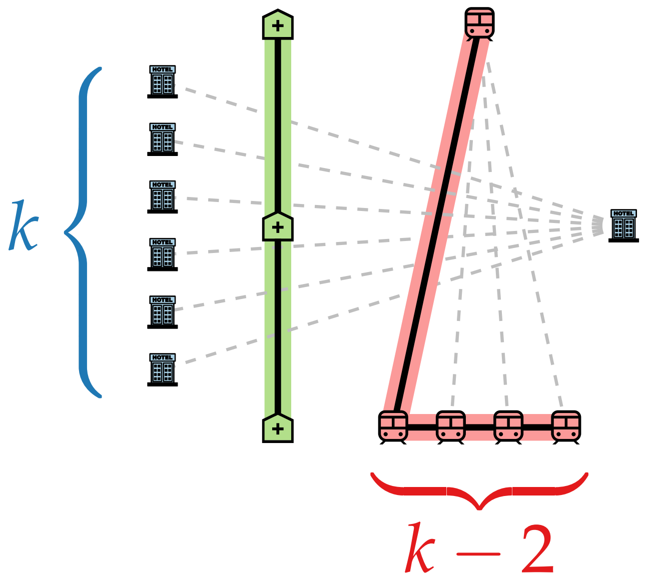


ReverseGreedy:

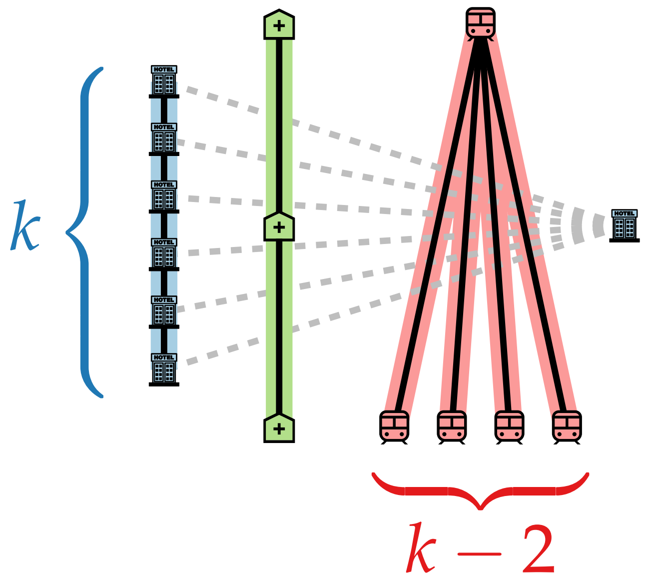


Bad Examples

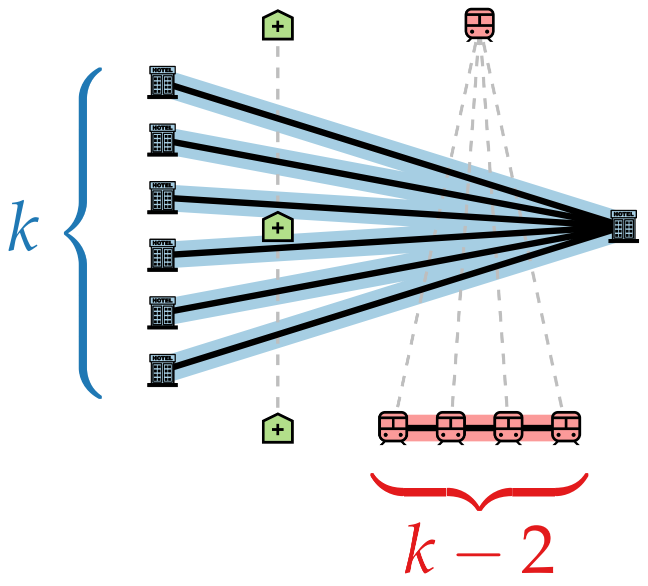
GREEDY:



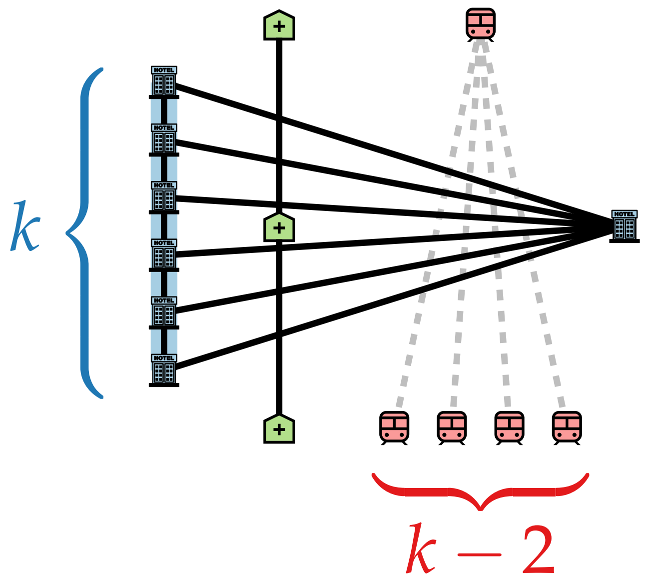
GREEDY:



REVERSEGREEDY:

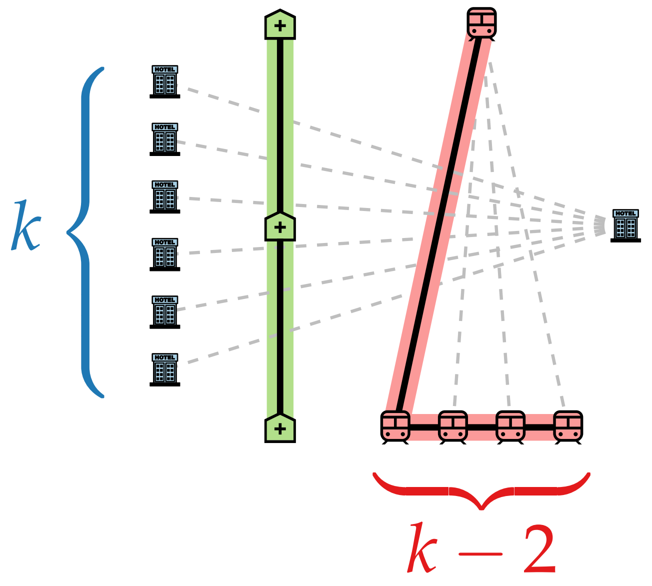


REVERSEGREEDY:

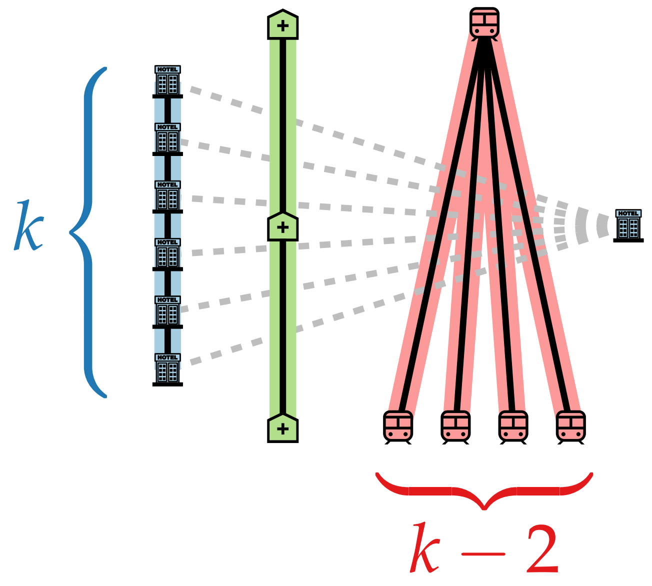


Bad Examples

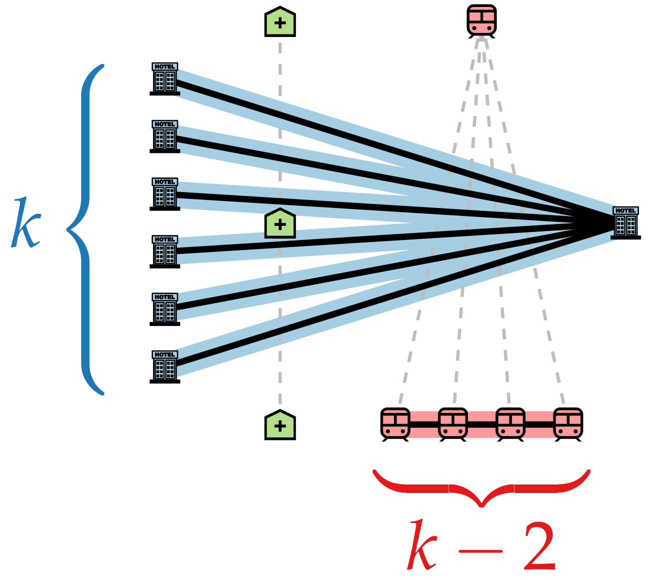
GREEDY:



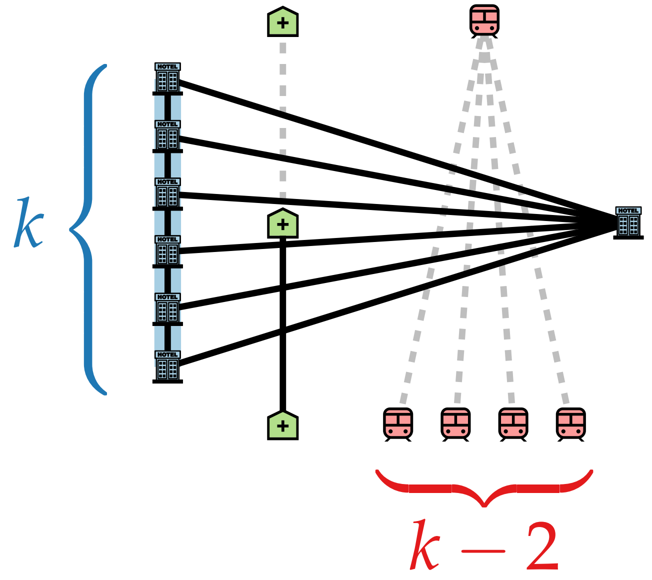
GREEDY:



REVERSEGREEDY:

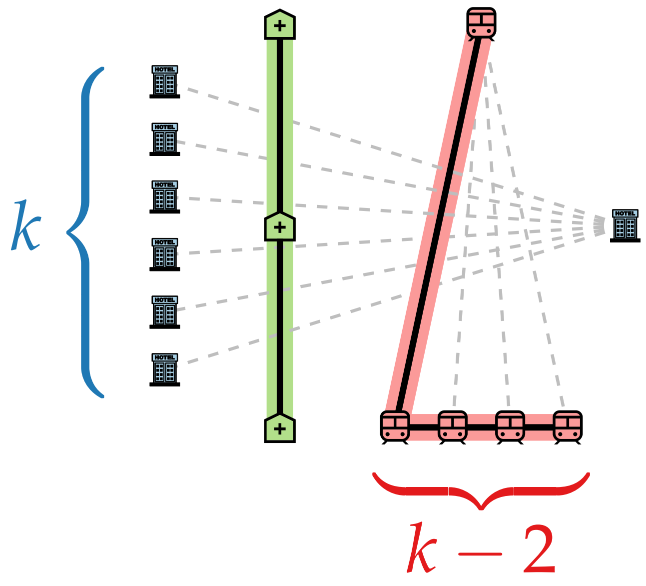


REVERSEGREEDY:

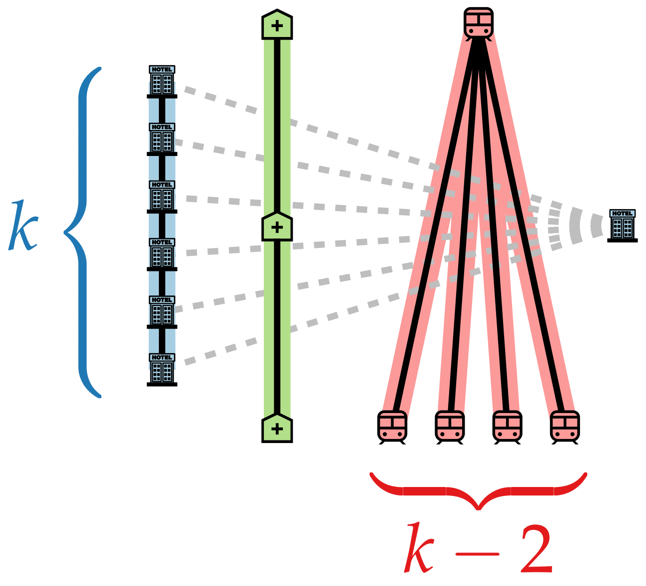


Bad Examples

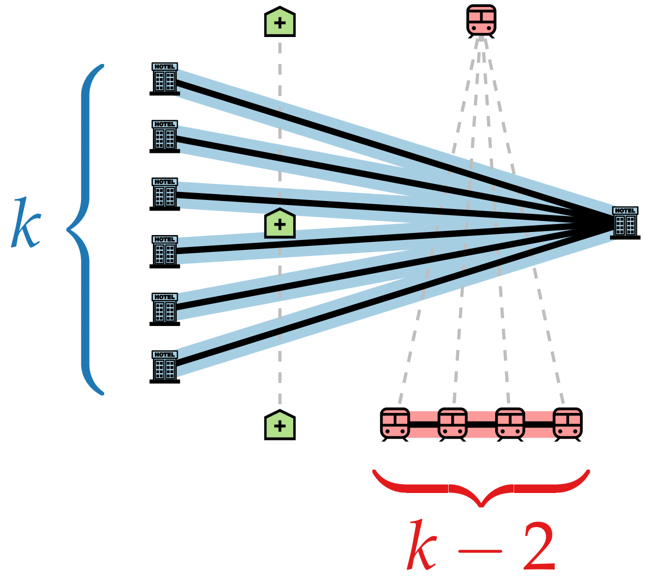
Greedy:



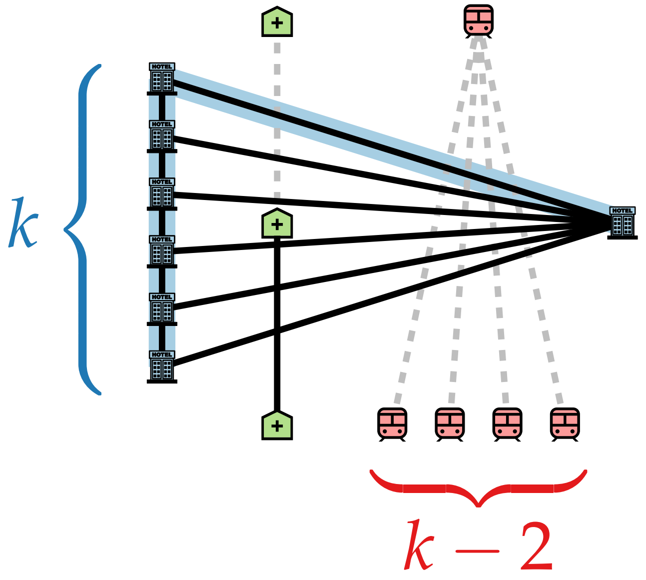
Greedy:



ReverseGreedy:

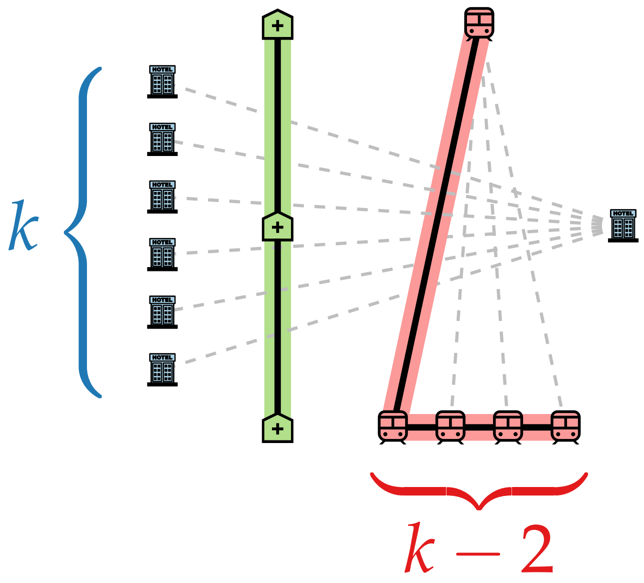


ReverseGreedy:

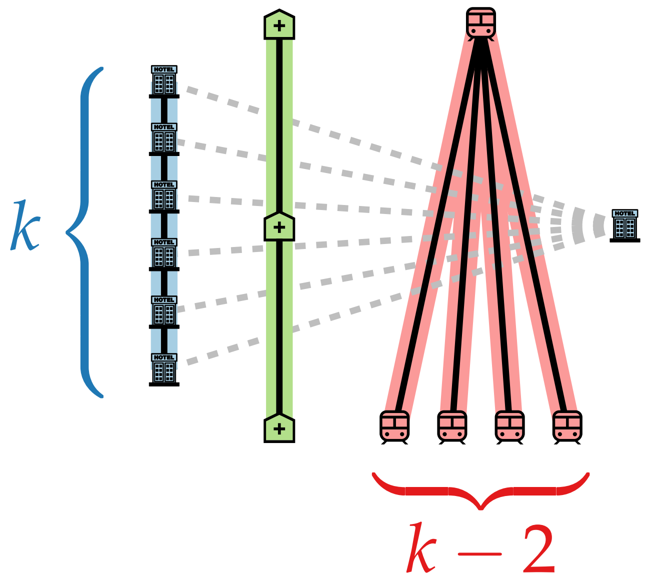


Bad Examples

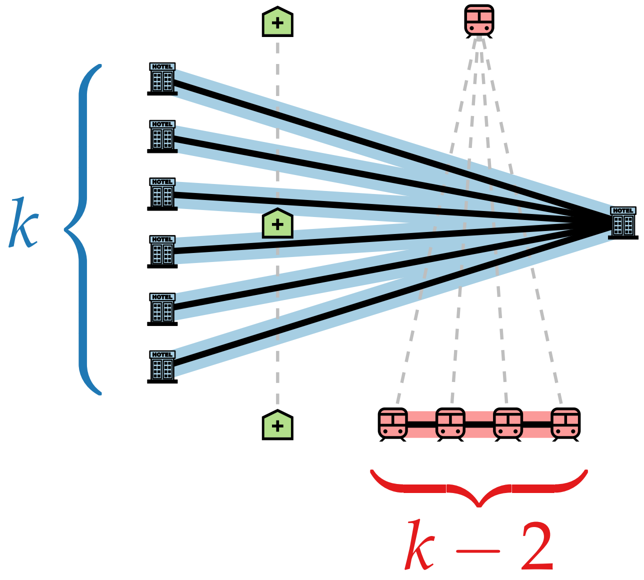
Greedy:



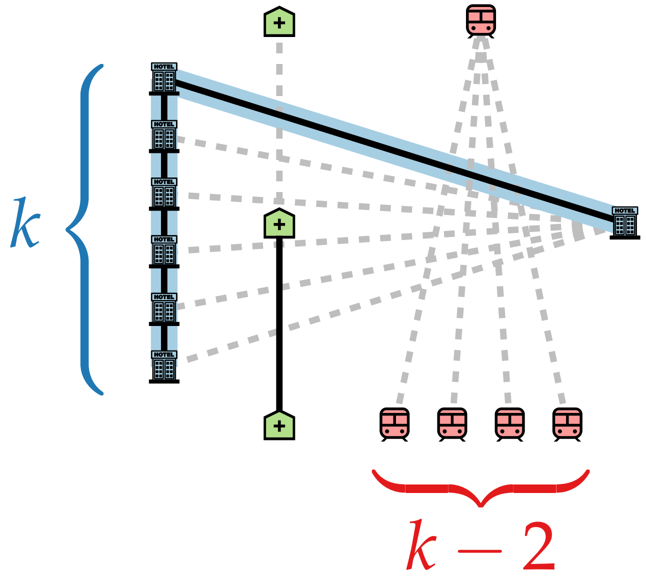
Greedy:



ReverseGreedy:

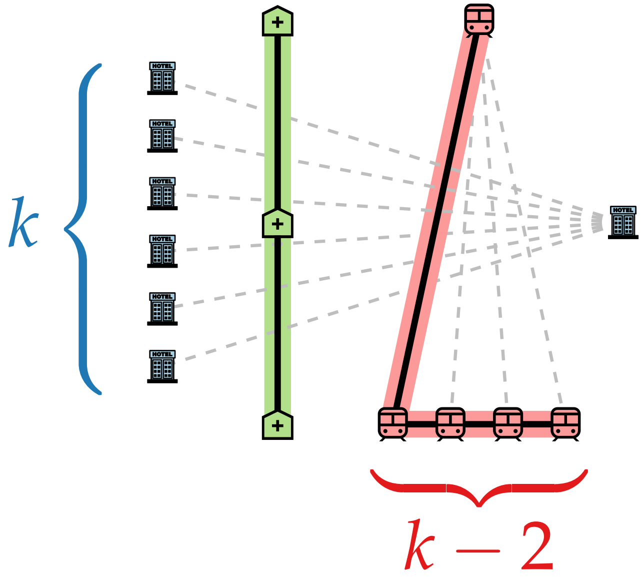


ReverseGreedy:

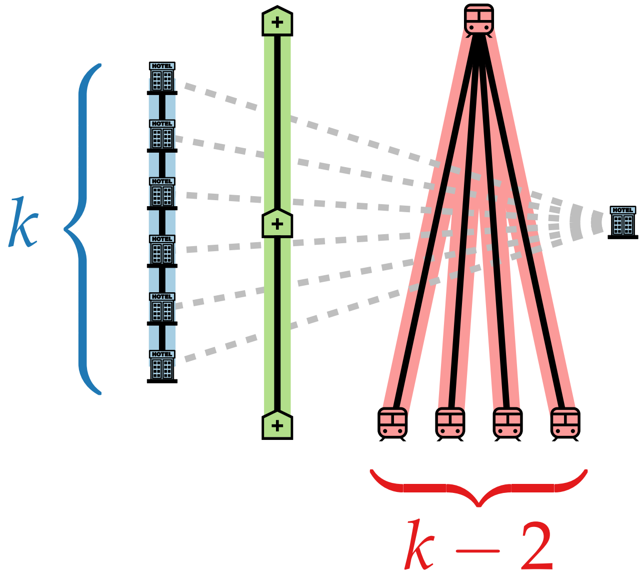


Bad Examples

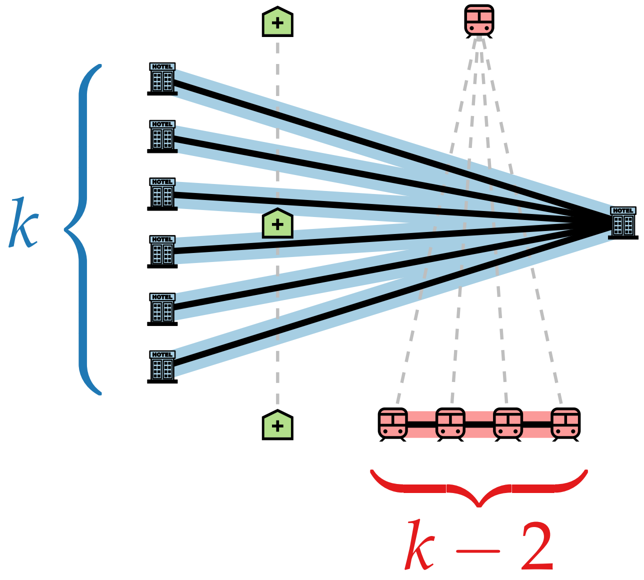
Greedy:



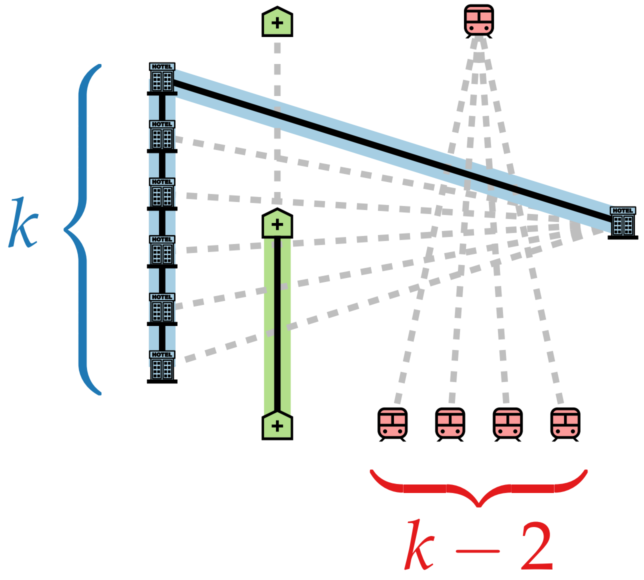
Greedy:



ReverseGreedy:



ReverseGreedy:



Planar Spanning Forest

- Greedy Heuristic

Planar Spanning Forest

- Greedy Heuristic
- ReverseGreedy Heuristic

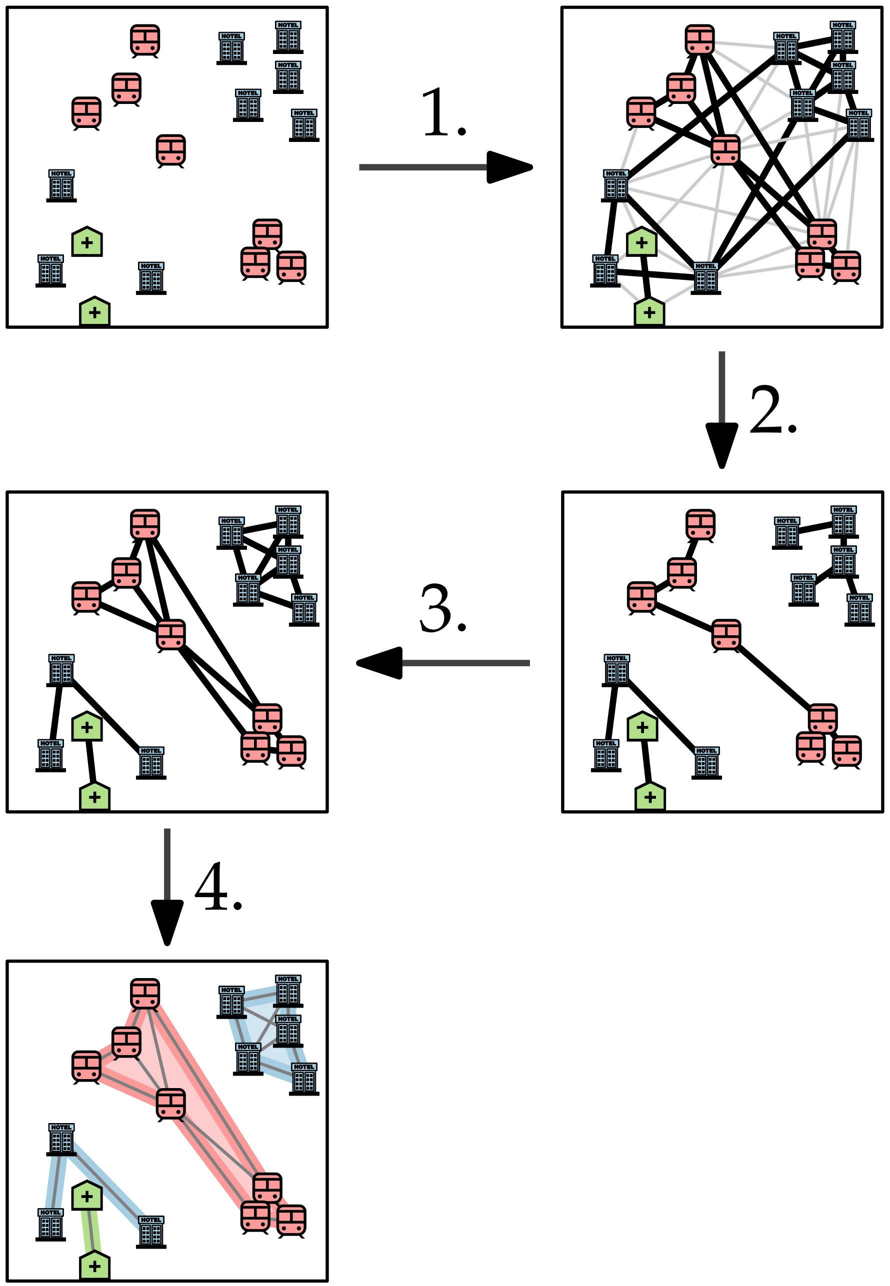
Planar Spanning Forest

- Greedy Heuristic
- ReverseGreedy Heuristic
- Exact ILP Formulation

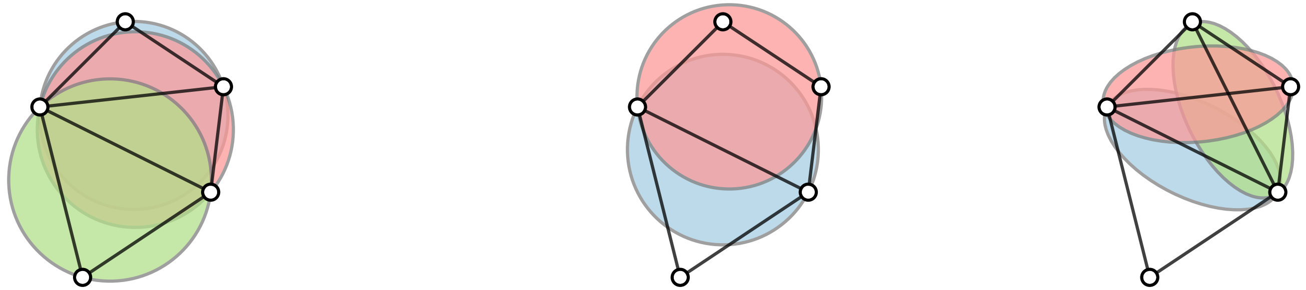
Planar Spanning Forest

- Greedy Heuristic
- ReverseGreedy Heuristic
- Exact ILP Formulation
- NP-hard even for one category [Jansen & Woeginger '93]

Pipeline



1. **Proximity Graph**

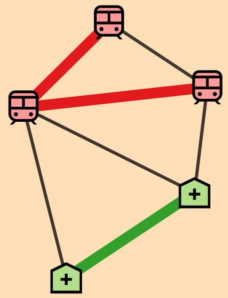


Delaunay Triangulation

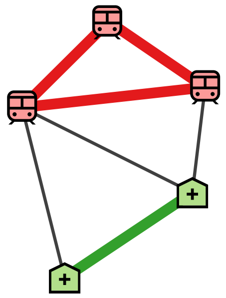
Gabriel Graph

β-Skeleton

2. **Planar Spanning Forest**

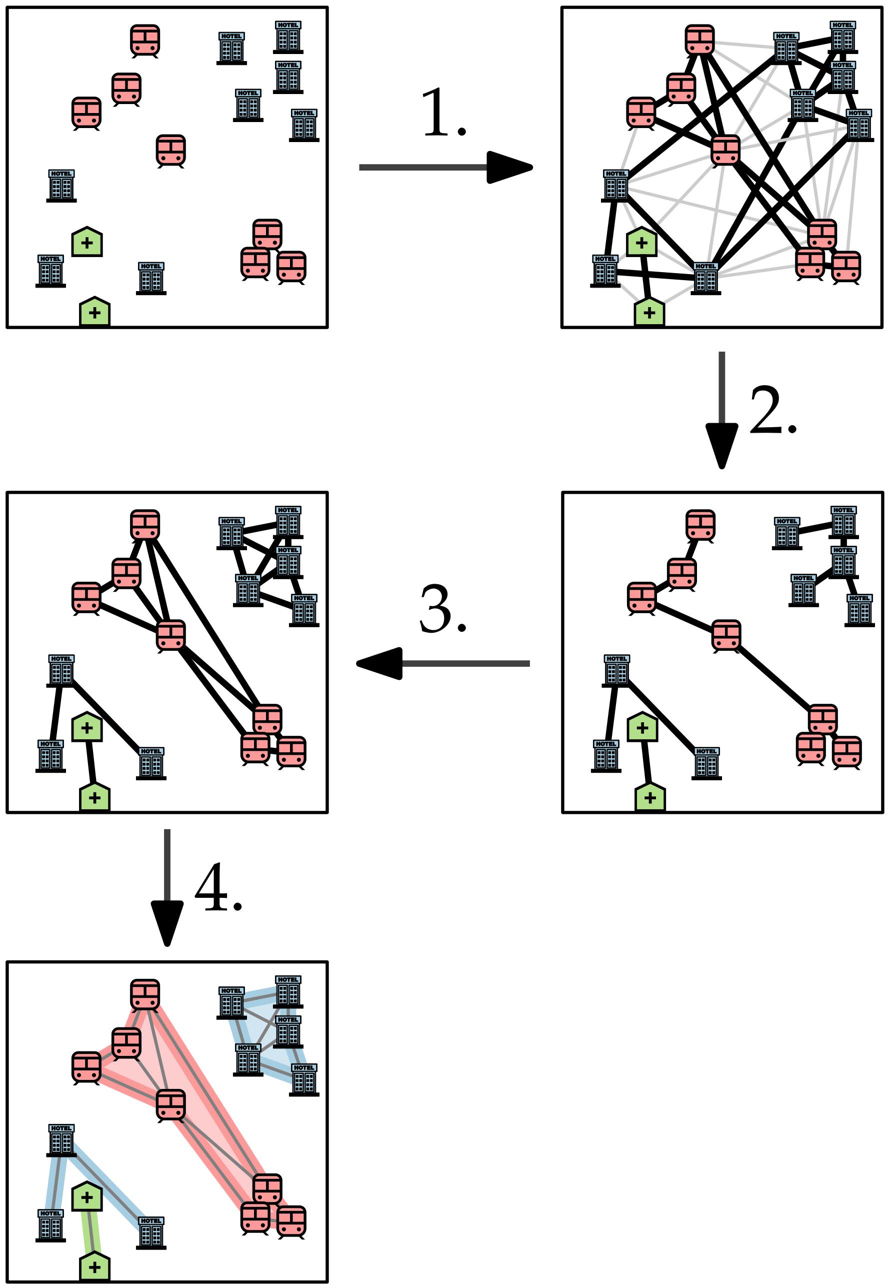


3. **Edge Augmentation**

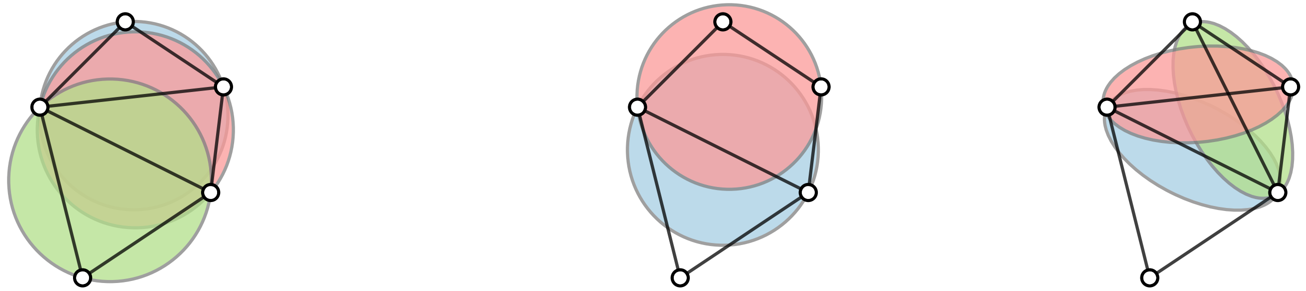


4. **Rendering**
   - Line Voronoi Diagram
   - Tree Representation
   - Polygon Representation

Pipeline



1. Proximity Graph

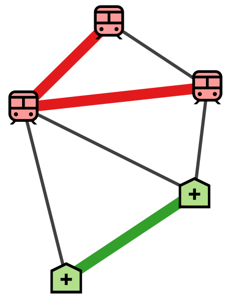


Delaunay Triangulation

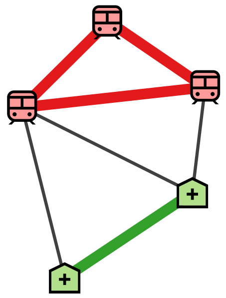
Gabriel Graph

β-Skeleton

2. Planar Spanning Forest



3. Edge Augmentation

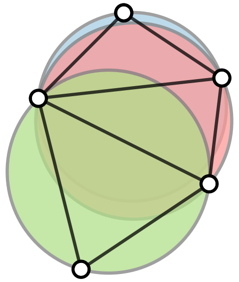


4. Rendering

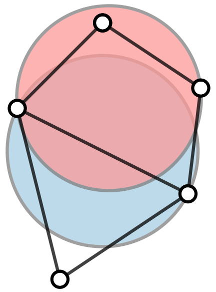
- Line Voronoi Diagram
- Tree Representation
- Polygon Representation

Proximity Graph

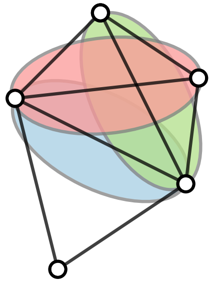
1. Proximity Graph



Delaunay Triangulation



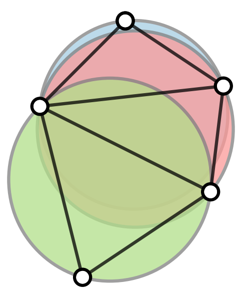
Gabriel Graph



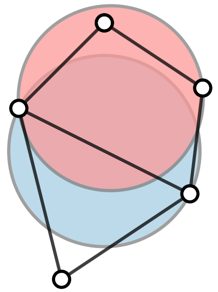
β-Skeleton

Proximity Graph

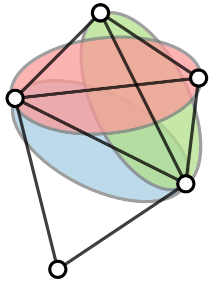
1. Proximity Graph



Delaunay Triangulation $\supseteq$ Gabriel Graph

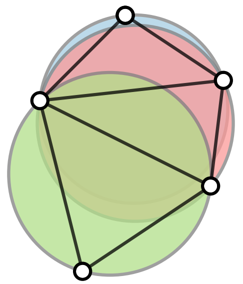


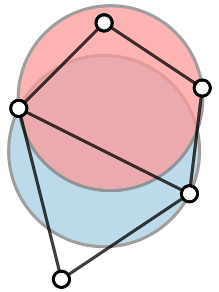
β-Skeleton

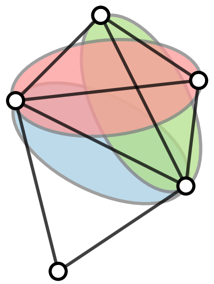


Proximity Graph

1. **Proximity Graph**





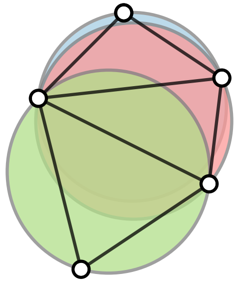


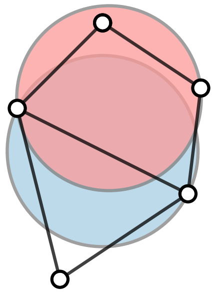
planar → Delaunay Triangulation $\supseteq$ Gabriel Graph

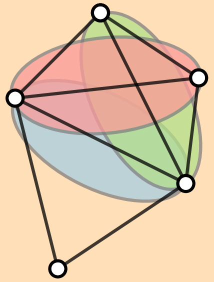
β-Skeleton

Proximity Graph

1. Proximity Graph





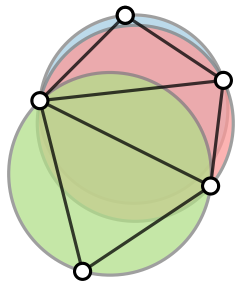


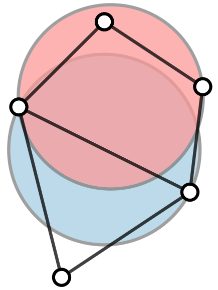
planar → Delaunay Triangulation ⊇ Gabriel Graph

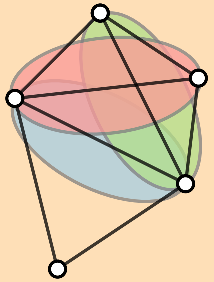
β-Skeleton

Proximity Graph

1. **Proximity Graph**

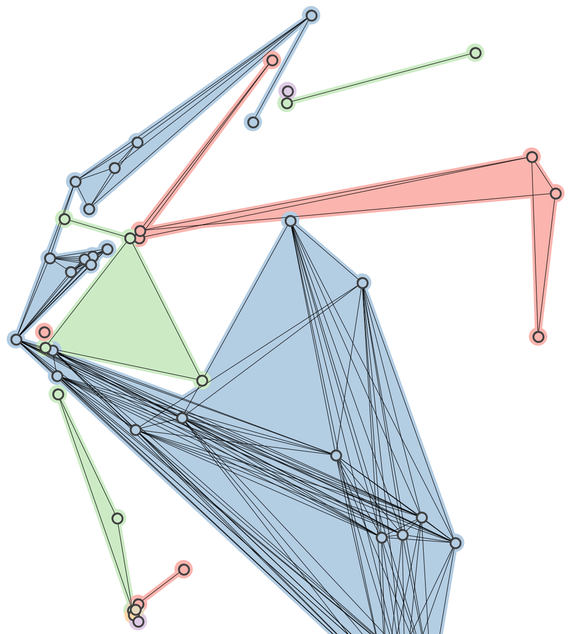






planar → Delaunay Triangulation ⊇ Gabriel Graph

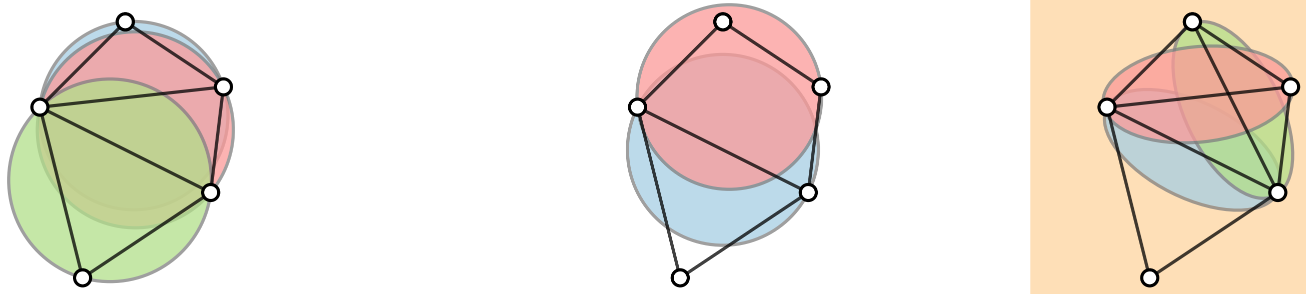
β-Skeleton



$\beta = 0.0$; 11 clusters

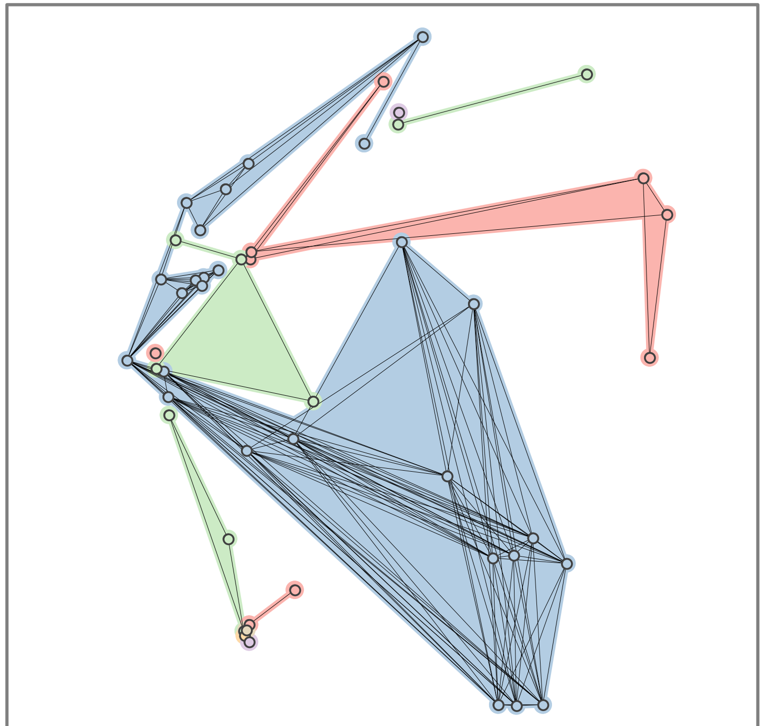
Proximity Graph

1. Proximity Graph

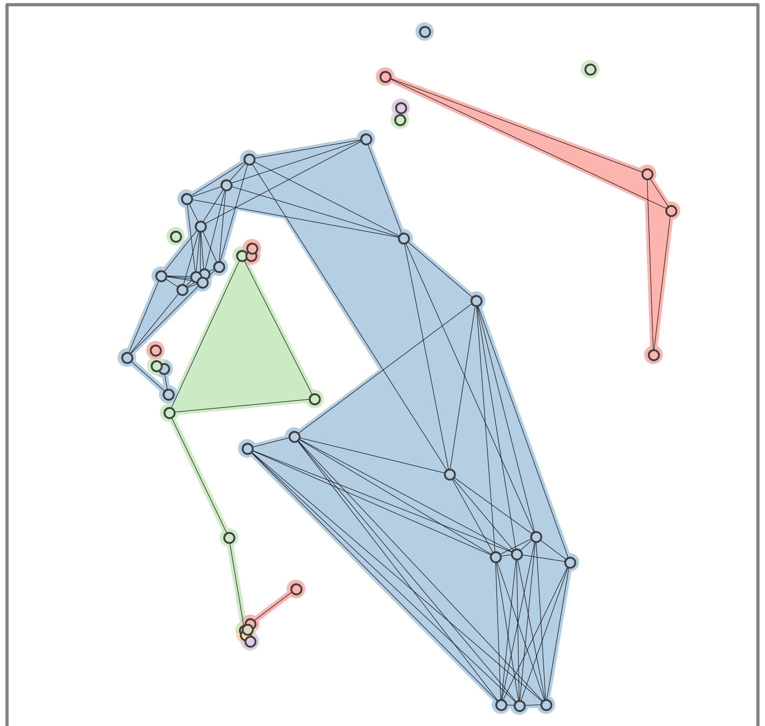


planar → Delaunay Triangulation ⊇ Gabriel Graph

β-Skeleton



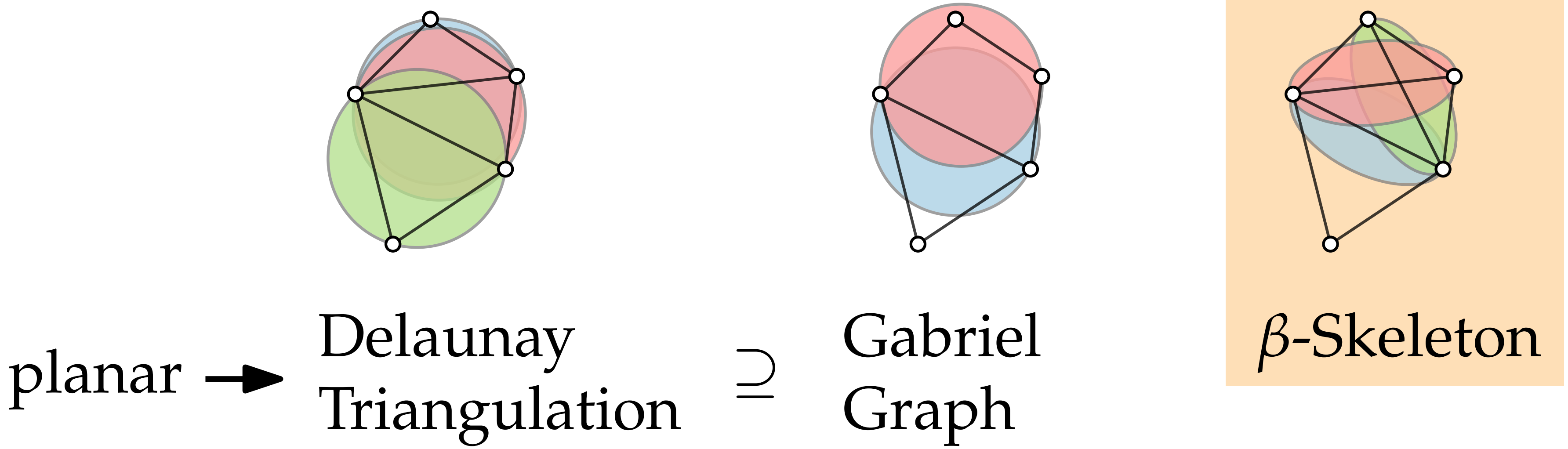
$\beta = 0.0$; 11 clusters

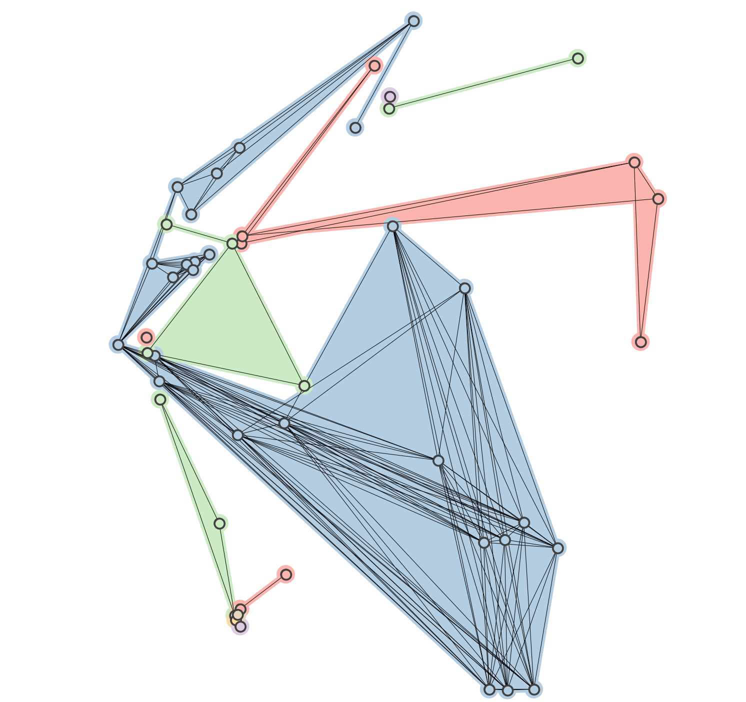


$\beta = 0.5$; 15 clusters

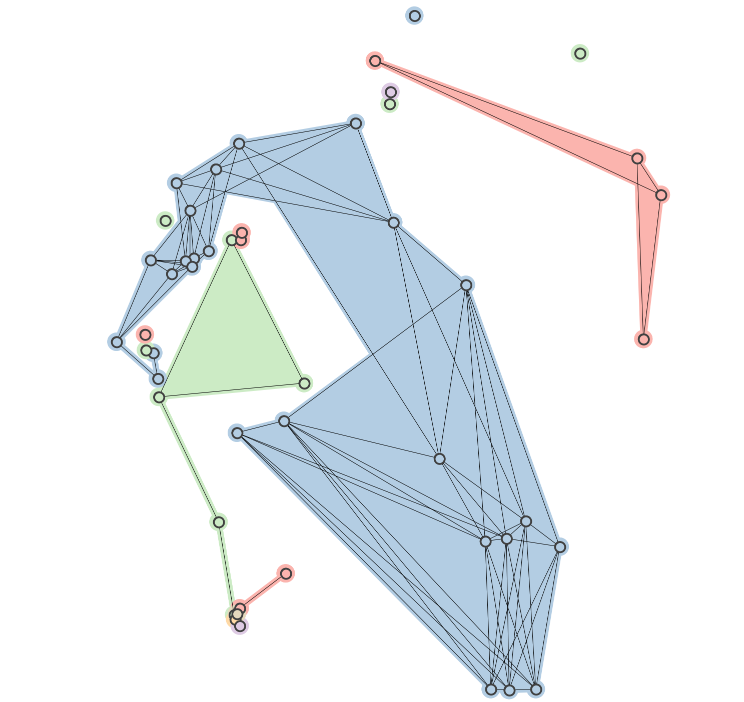
Proximity Graph

1. **Proximity Graph**

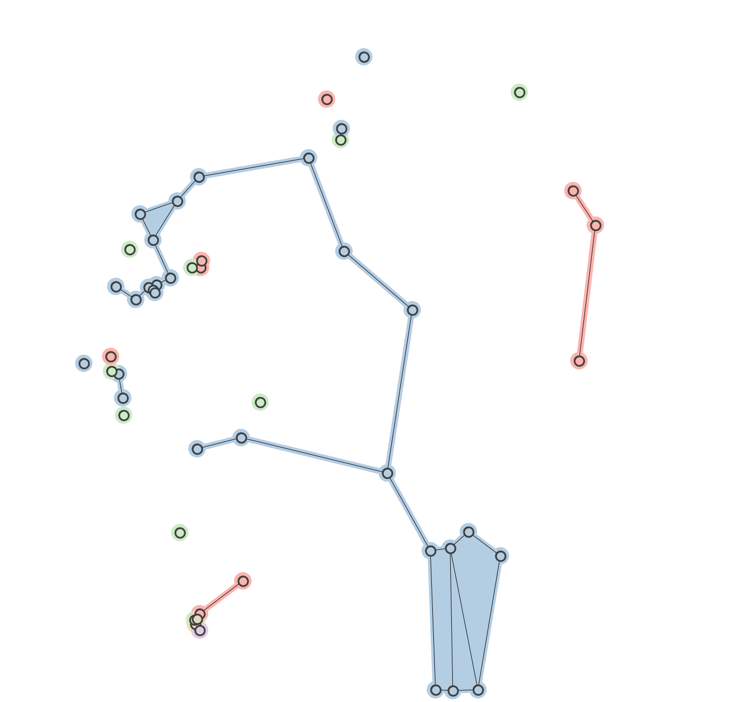




$\beta = 0.0$; 11 clusters

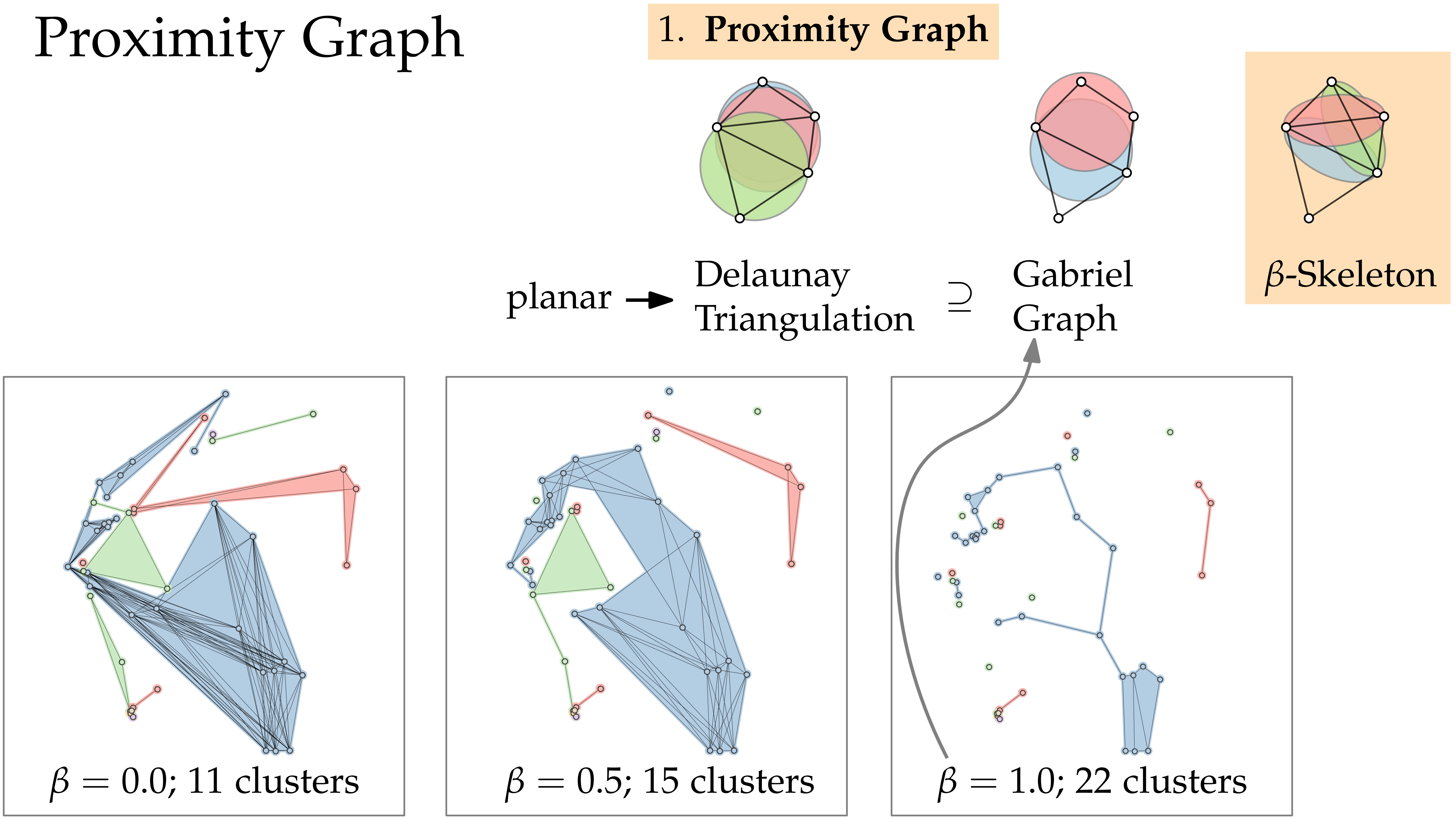


$\beta = 0.5$; 15 clusters

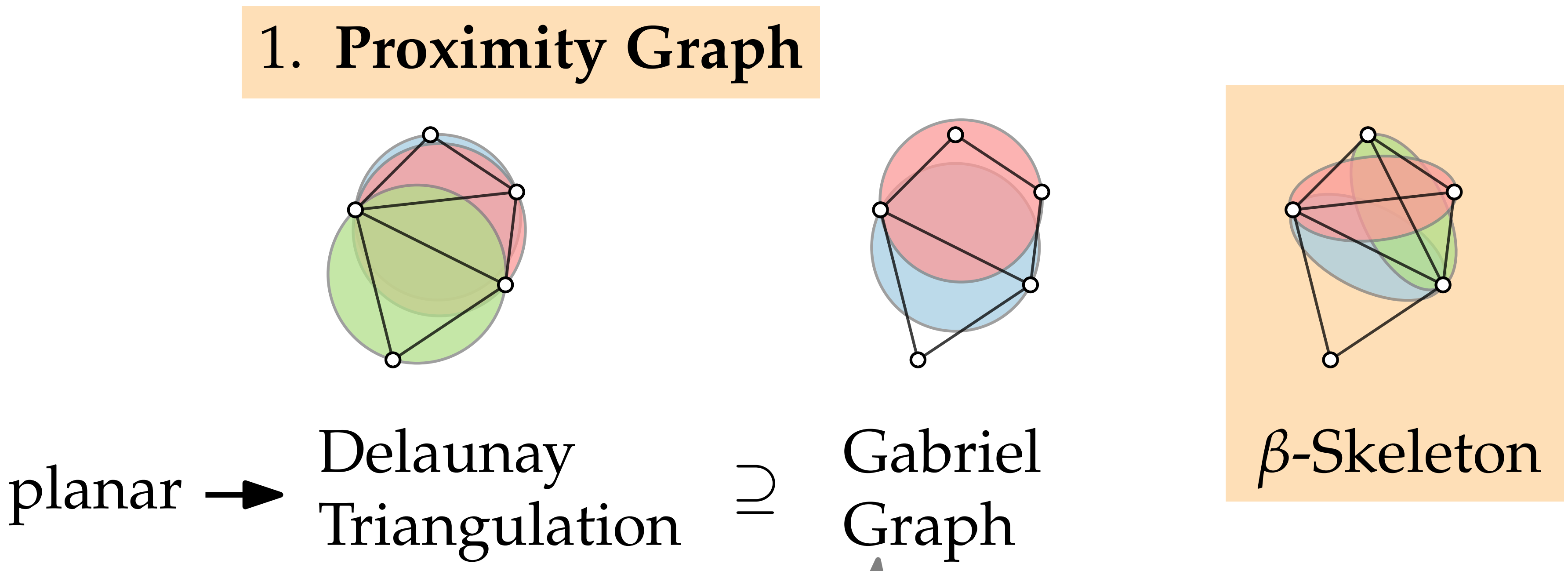


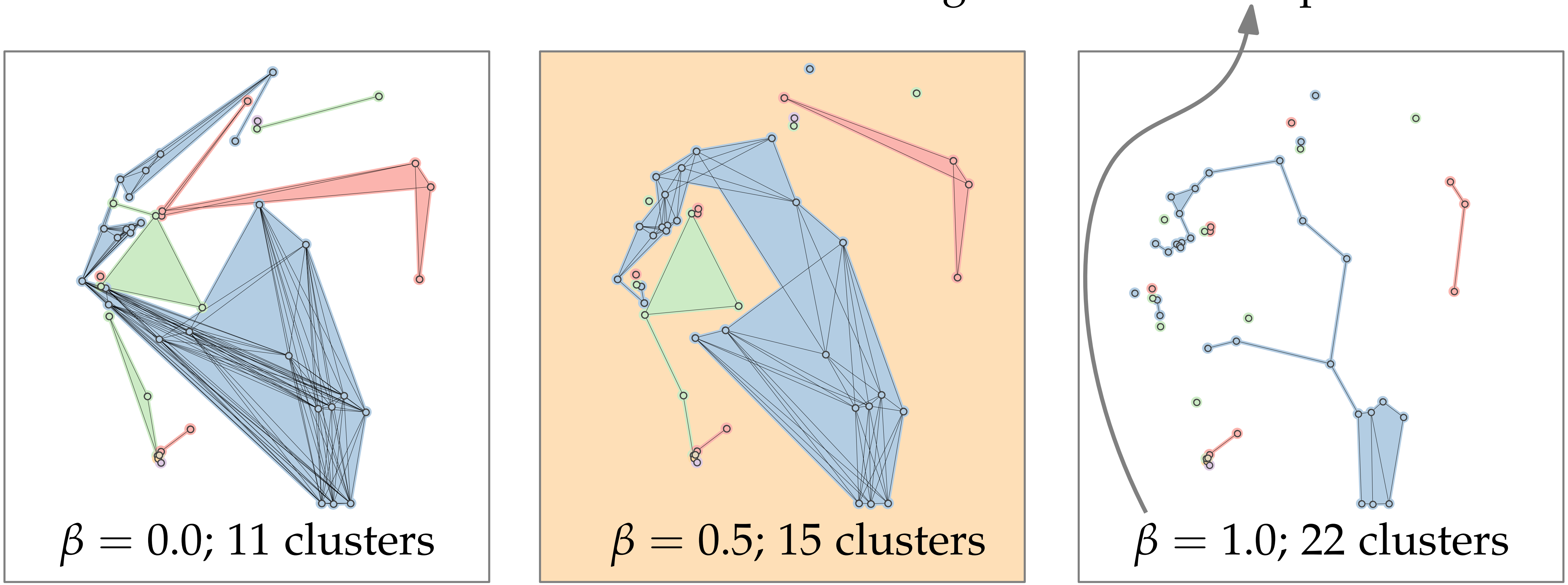
$\beta = 1.0$; 22 clusters

Proximity Graph



Proximity Graph





Case Study

Case Study

UBN: University of Bonn, 78 points

Case Study

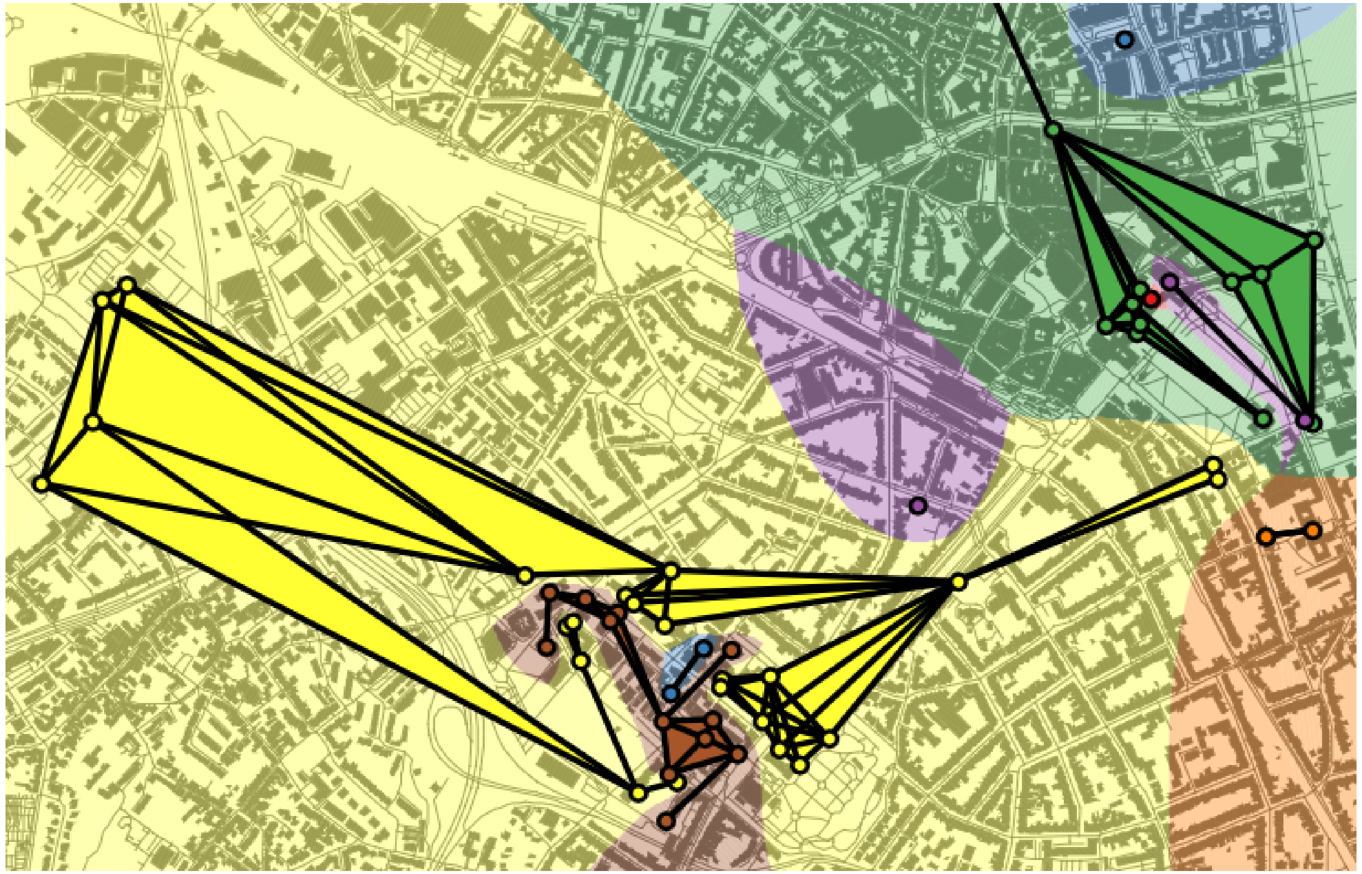
UBN: University of Bonn, 78 points

Voronoi Diagram

Case Study

UBN: University of Bonn, 78 points

Voronoi Diagram



Greedy Algorithm
0.5-skeleton
Line Voronoi Diagram
+ Polygon Representation

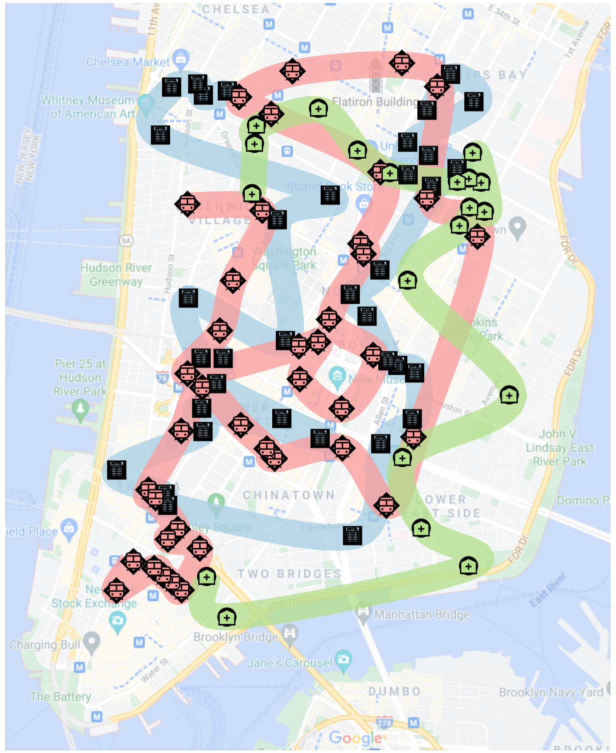
Case Study

NYC: Manhattan, New York, 96 points

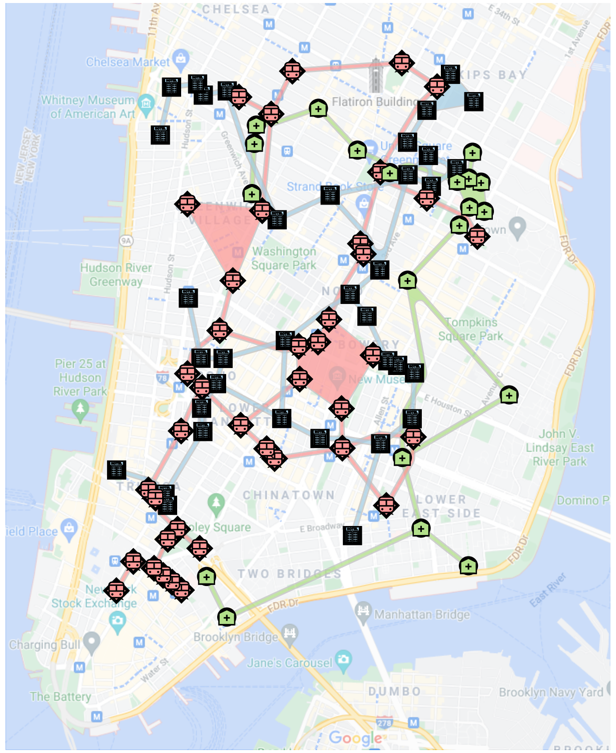
Case Study

NYC: Manhattan, New York, 96 points

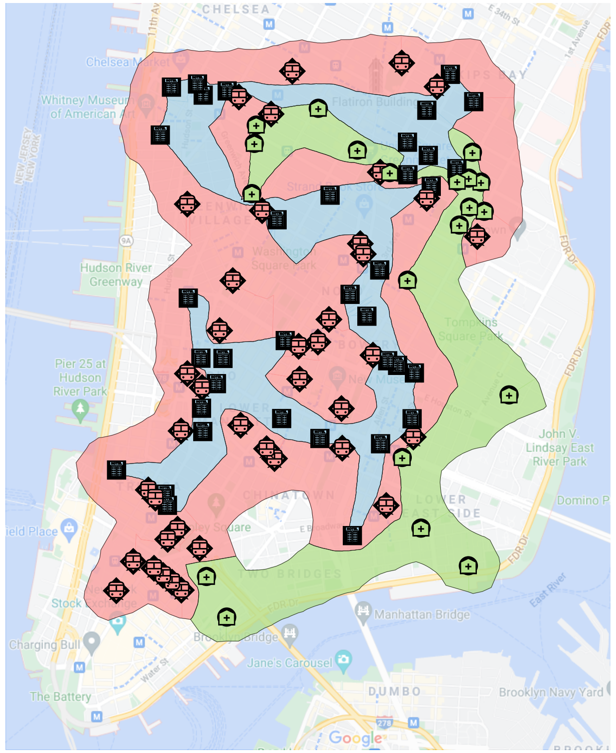
Bubble Sets



LineSets

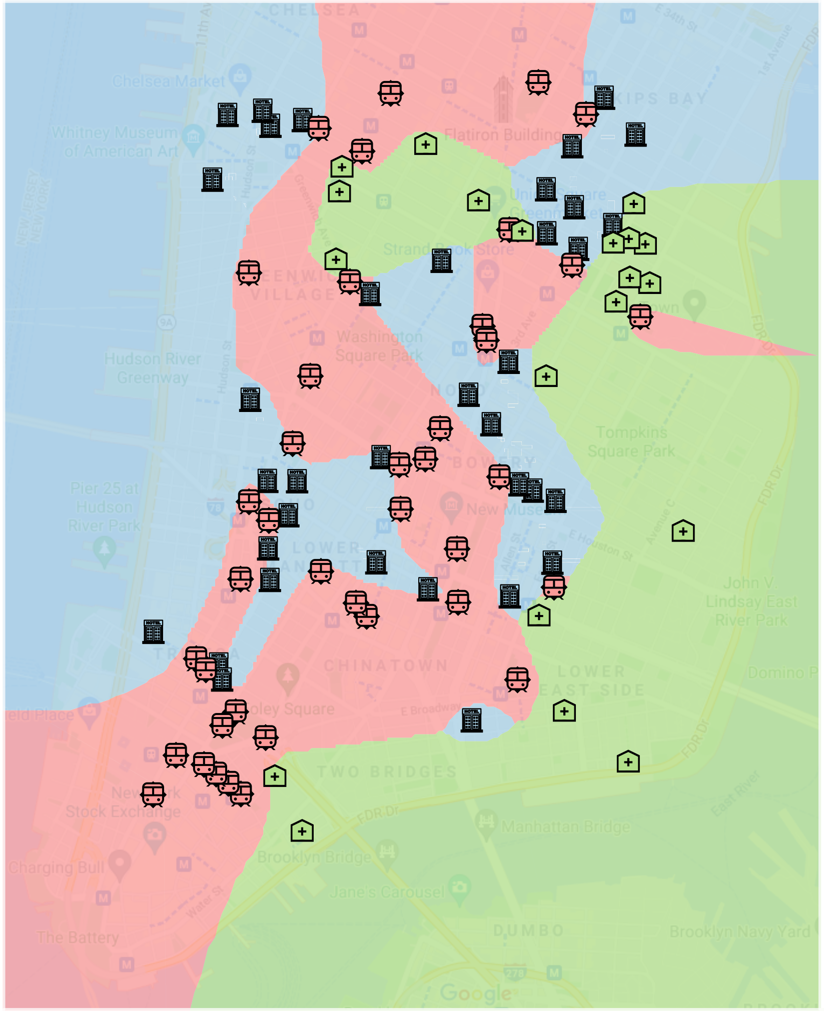


KelpFusion

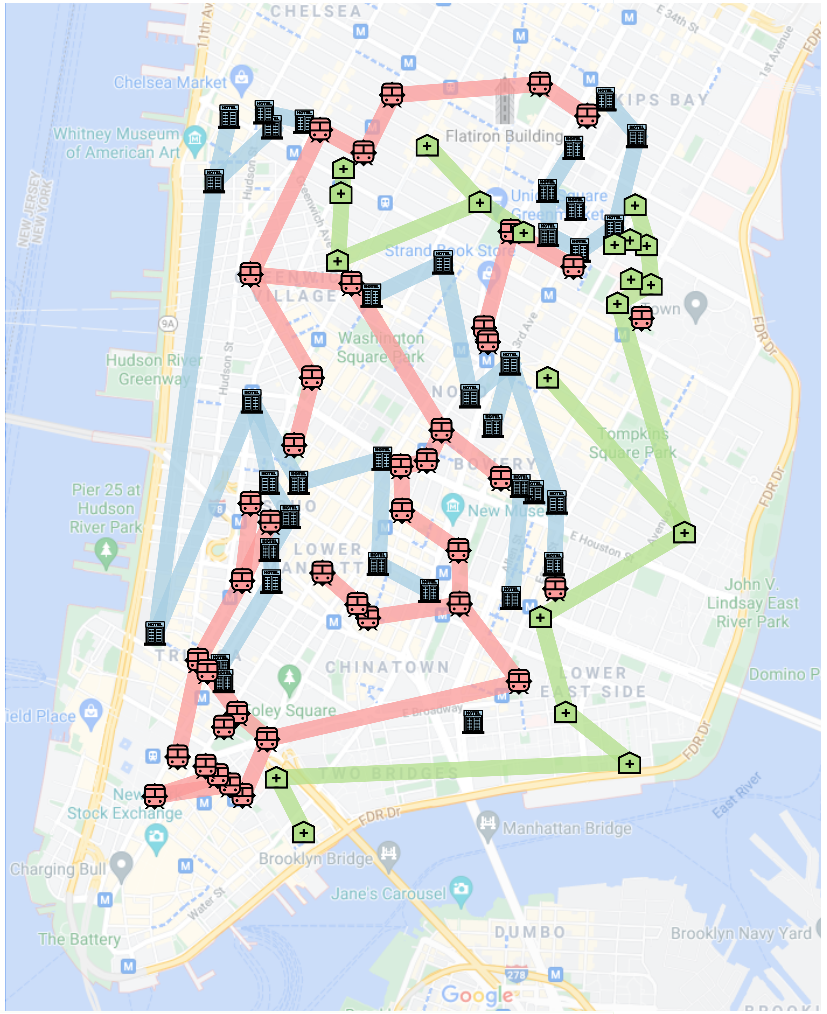


MapSets

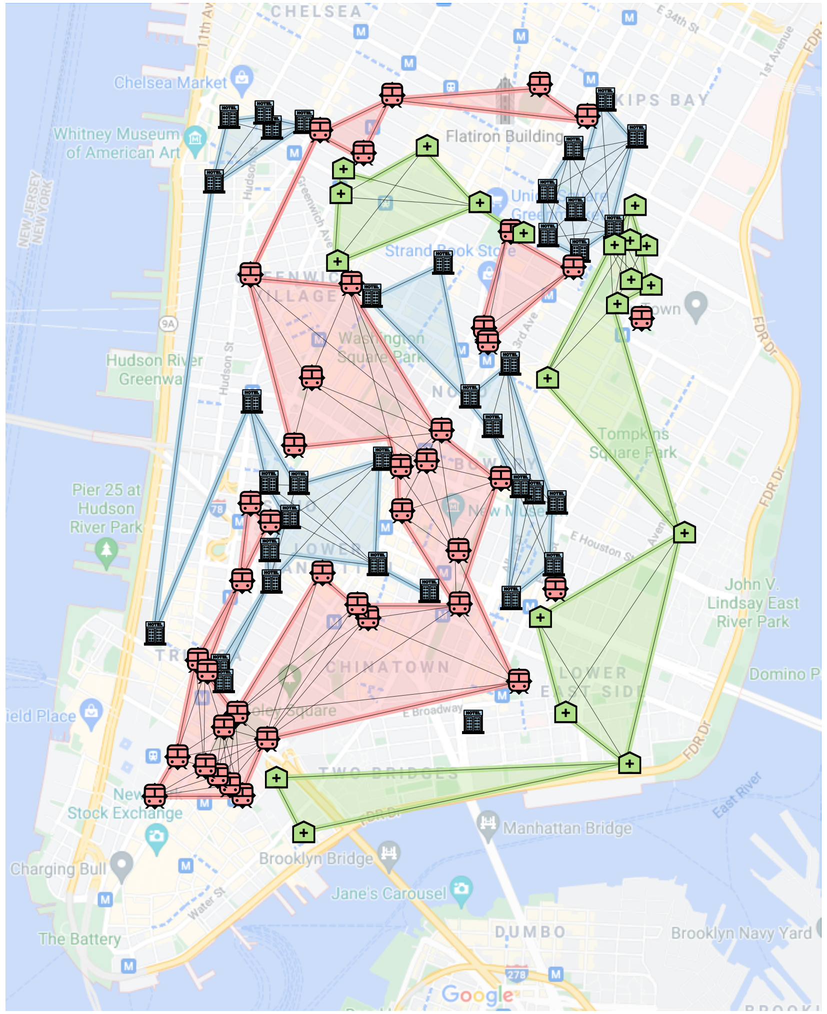
NYC: Manhattan, New York, 96 points



ClusterSets:
Line-Voronoi



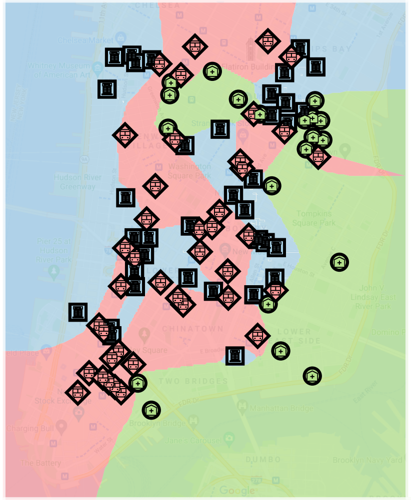
ClusterSets:
Tree Repr.



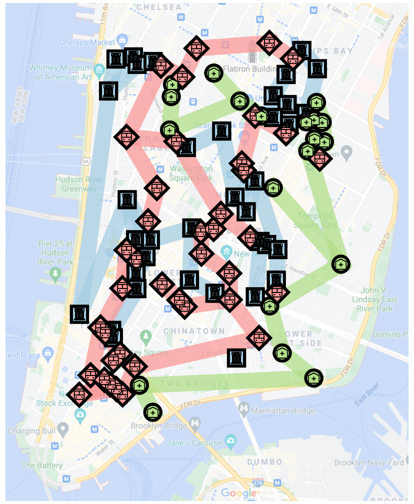
ClusterSets:
Polygon Repr.

Case Study

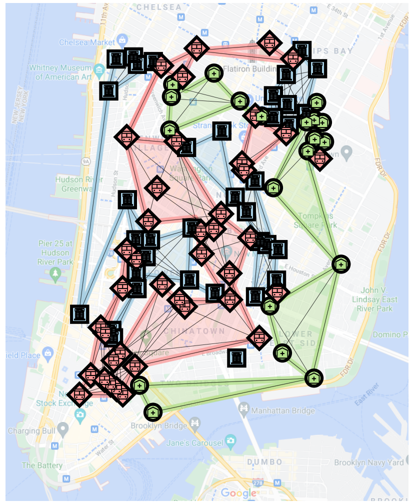
NYC: Manhattan, New York, 96 points



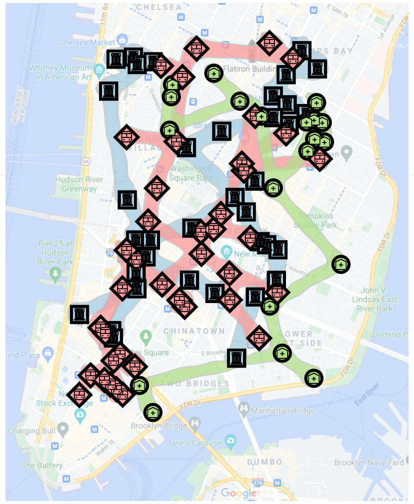
ClusterSets:
Line-Voronoi



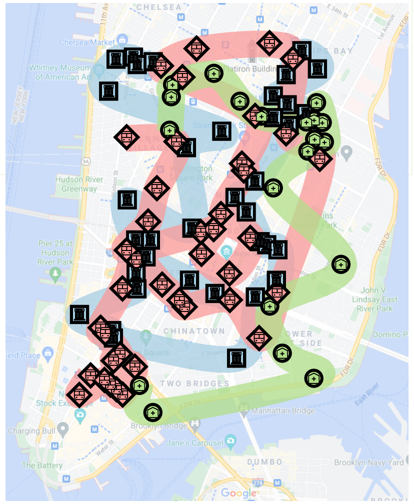
ClusterSets:
Tree Repr.



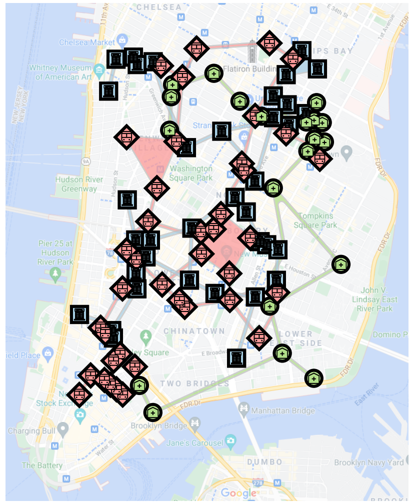
ClusterSets:
Polygon Repr.



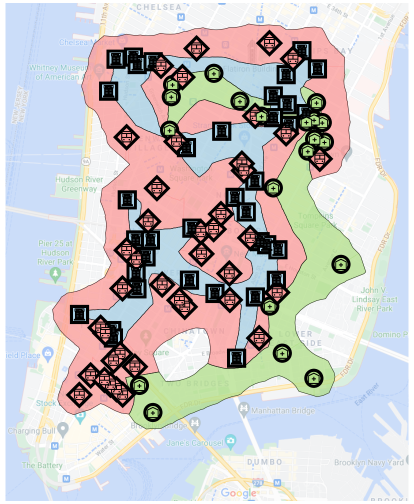
Bubble Sets



LineSets



KelpFusion



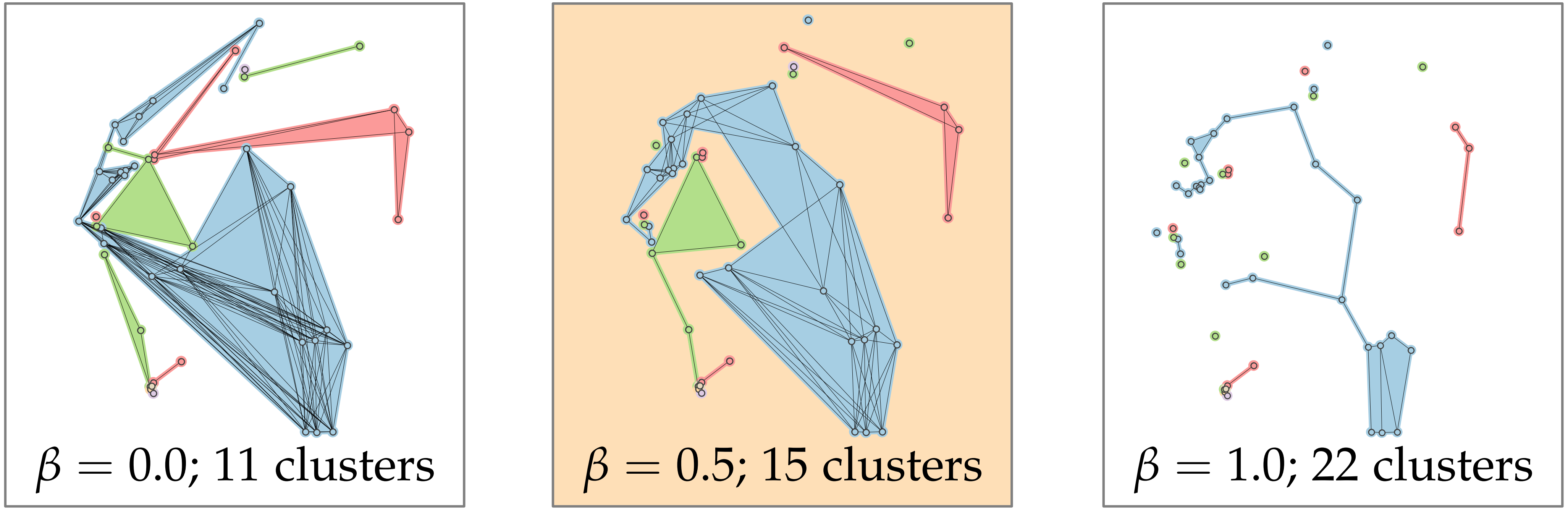
MapSets

Case Study

OSM: City of Bonn, 16320 points (OpenStreetMap)

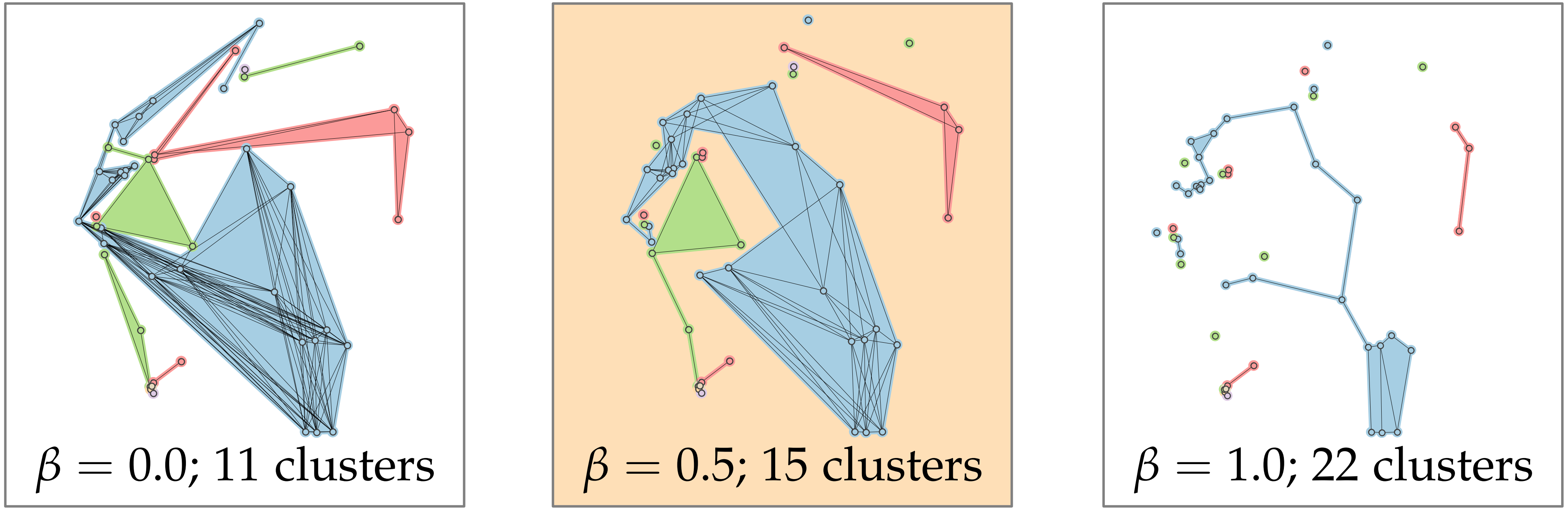
Case Study

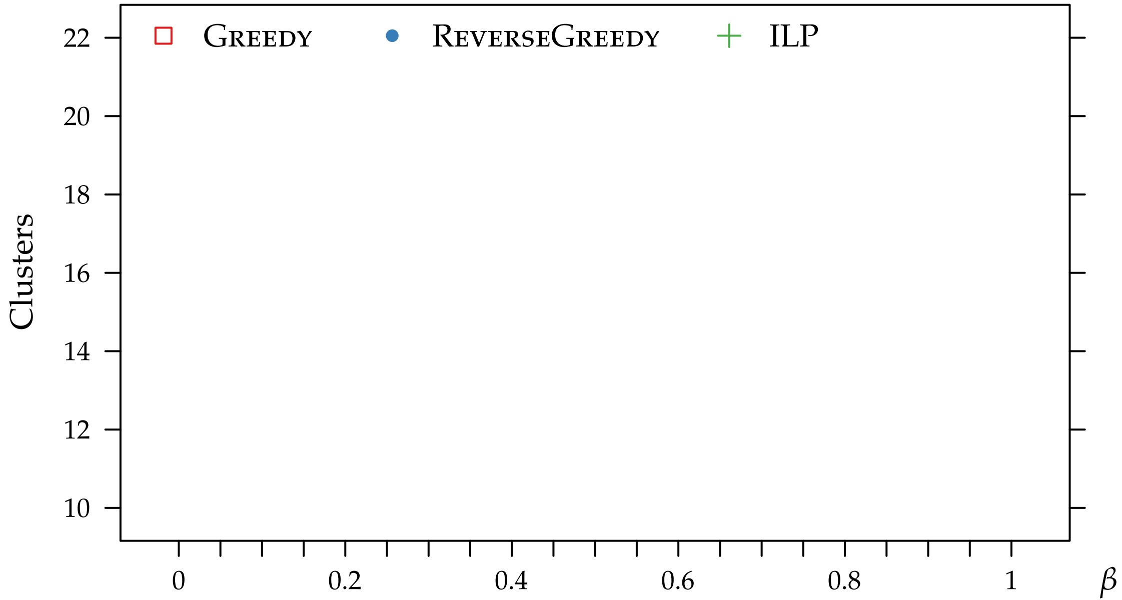
OSM: City of Bonn, 16320 points (OpenStreetMap)



Case Study

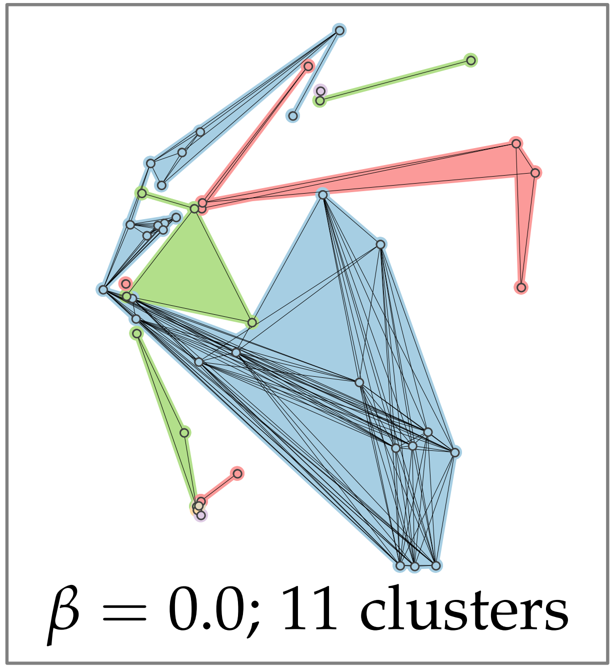
OSM: City of Bonn, 16320 points (OpenStreetMap)

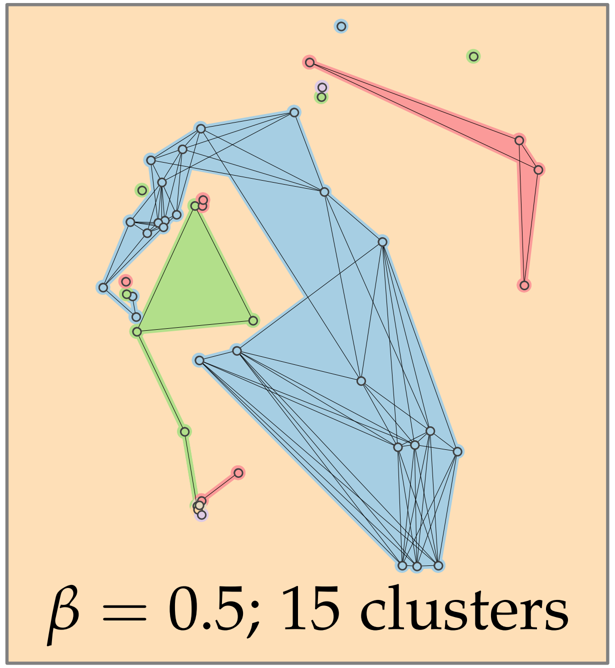


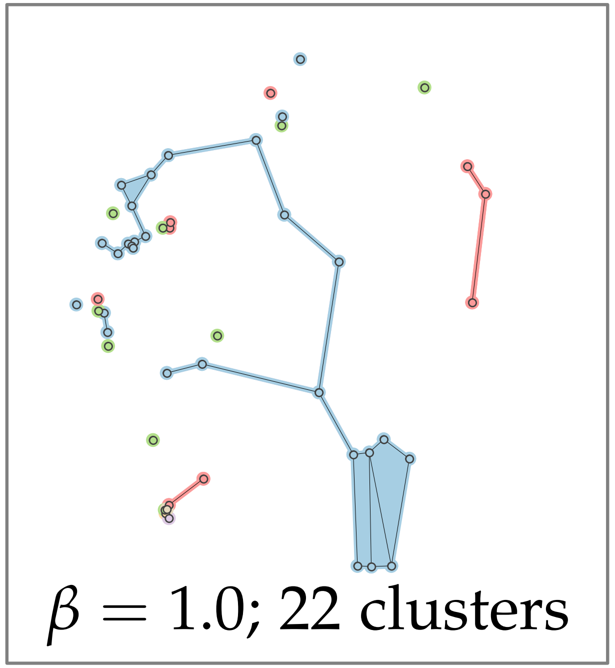


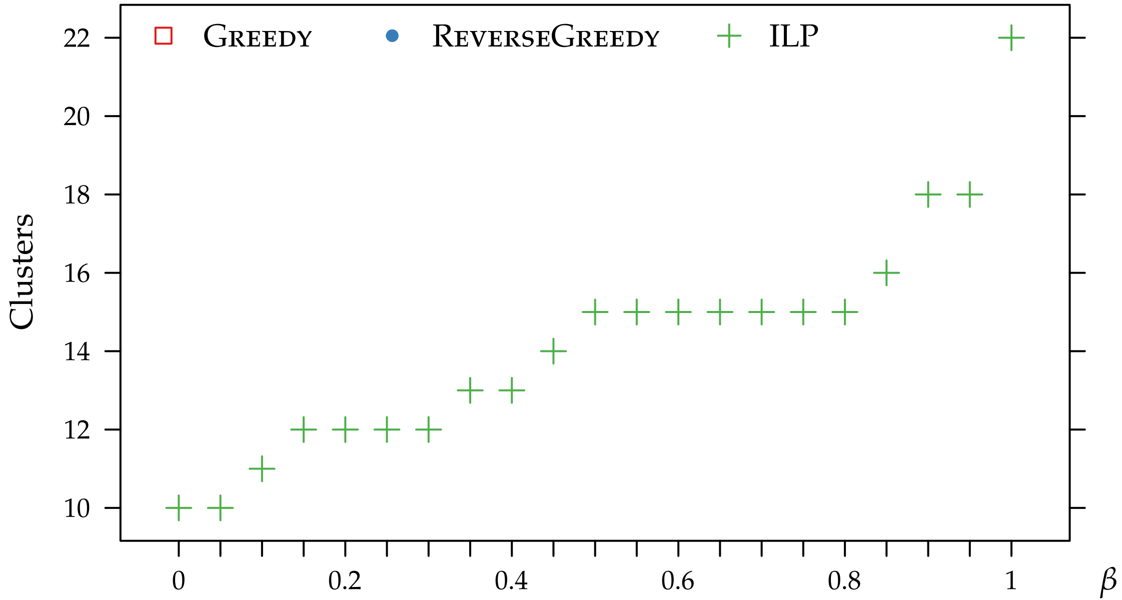
Case Study

OSM: City of Bonn, 16320 points (OpenStreetMap)



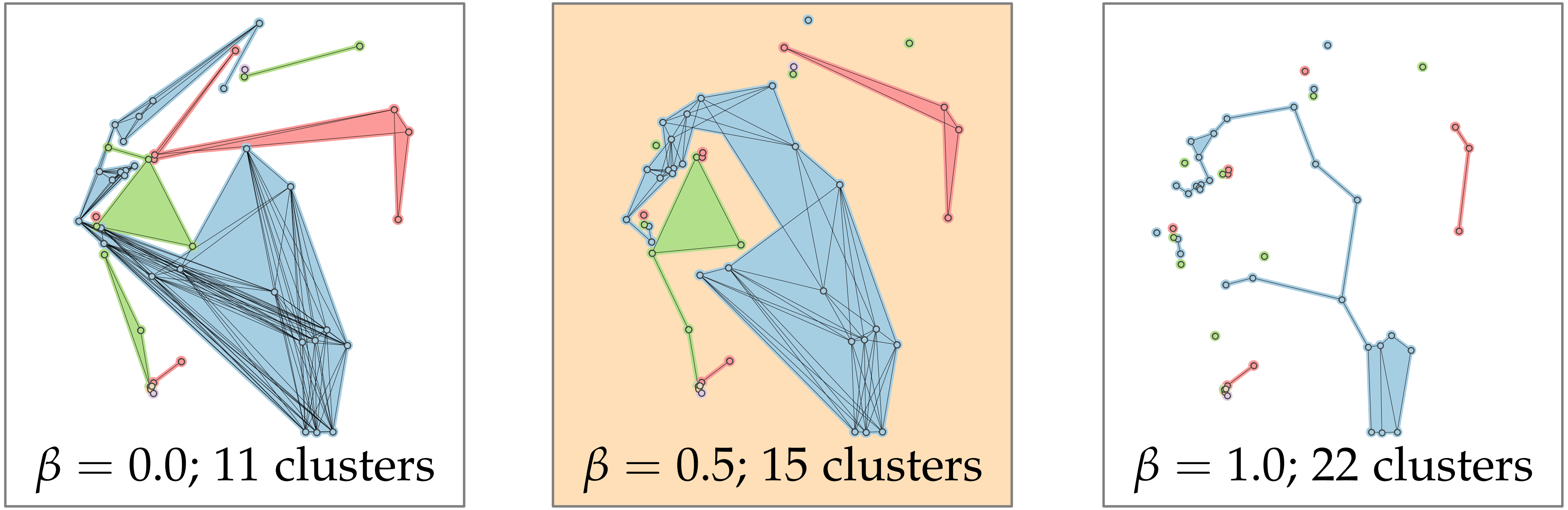


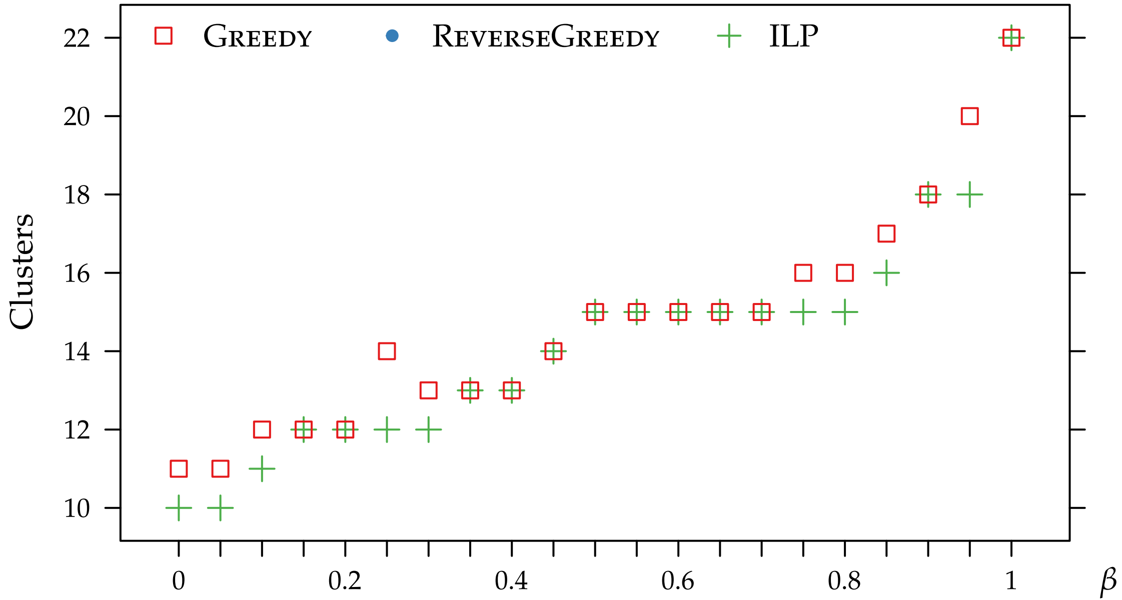




Case Study

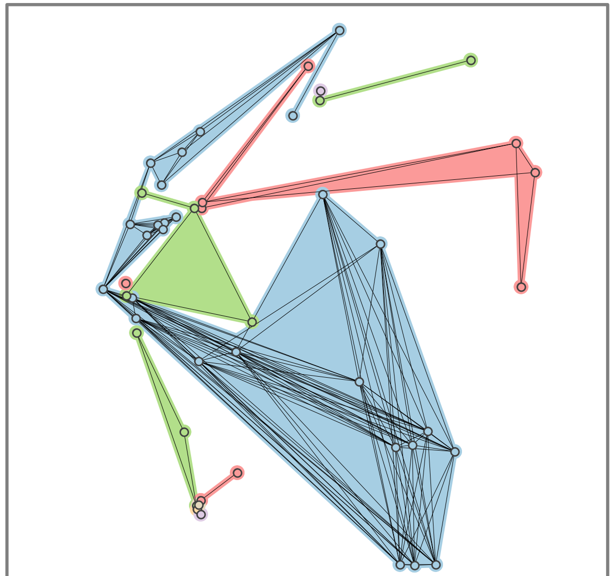
OSM: City of Bonn, 16320 points (OpenStreetMap)



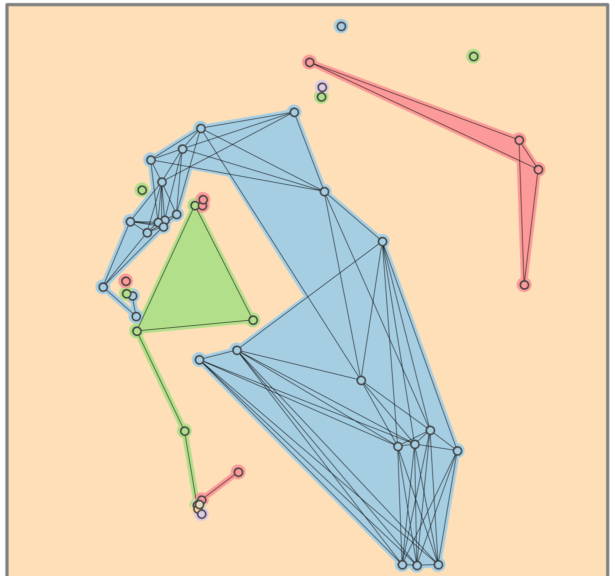


Case Study

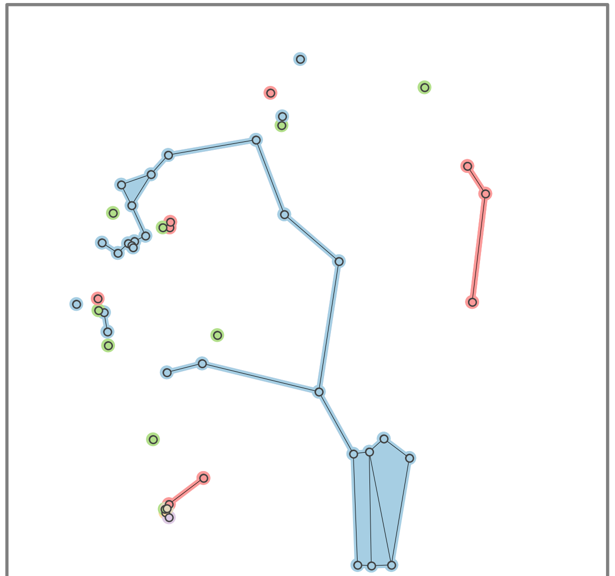
OSM: City of Bonn, 16320 points (OpenStreetMap)



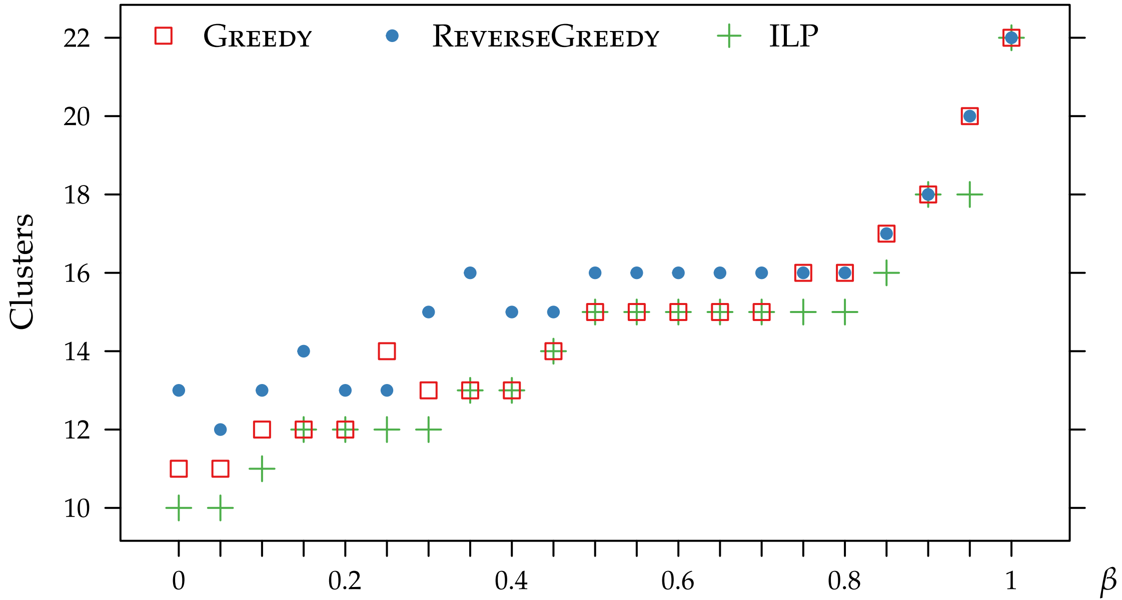
$\beta = 0.0$; 11 clusters



$\beta = 0.5$; 15 clusters



$\beta = 1.0$; 22 clusters



Quantitative Experiments

Quantitative Experiments

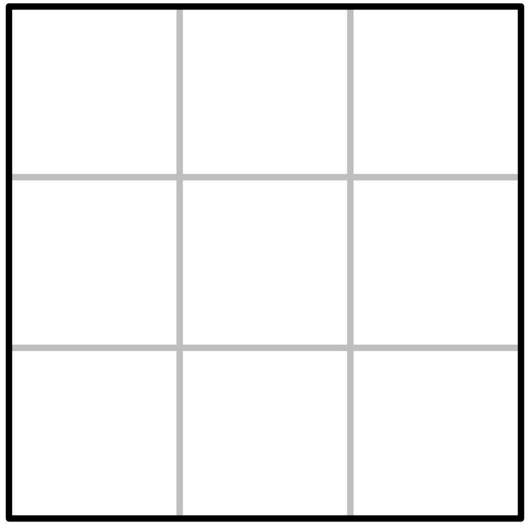
- Based on **OSM**

Quantitative Experiments

- Based on **OSM**

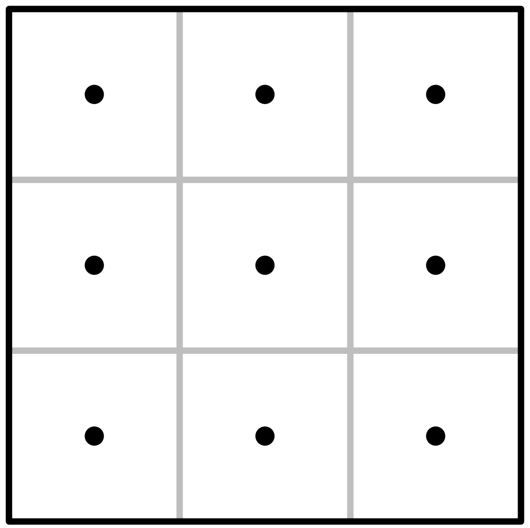
Quantitative Experiments

- Based on **OSM**

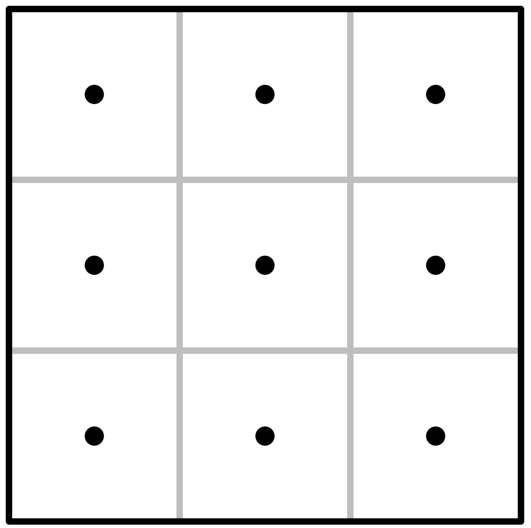


Quantitative Experiments

- Based on **OSM**

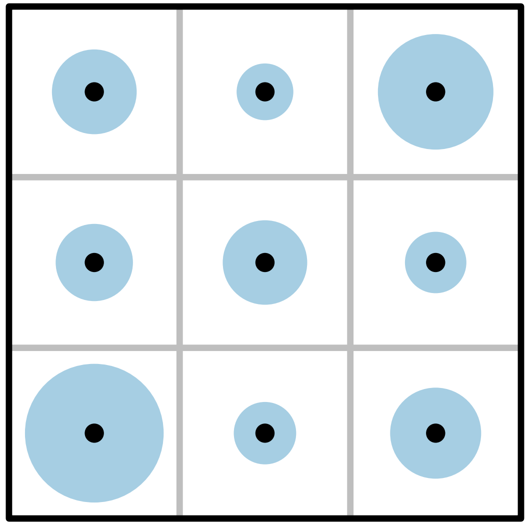


Quantitative Experiments



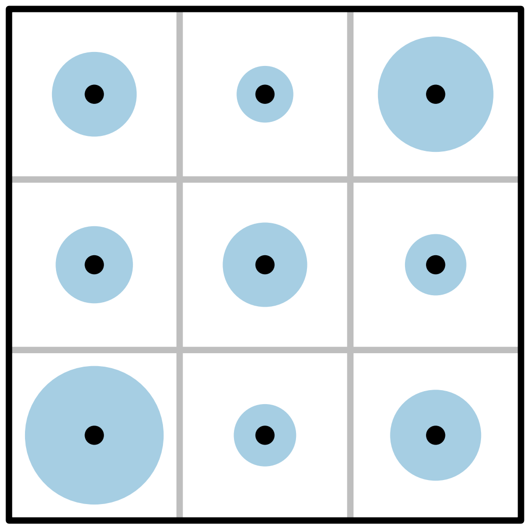
- Based on **OSM**
- $n = 50, 100, 150, 200, 250$

Quantitative Experiments



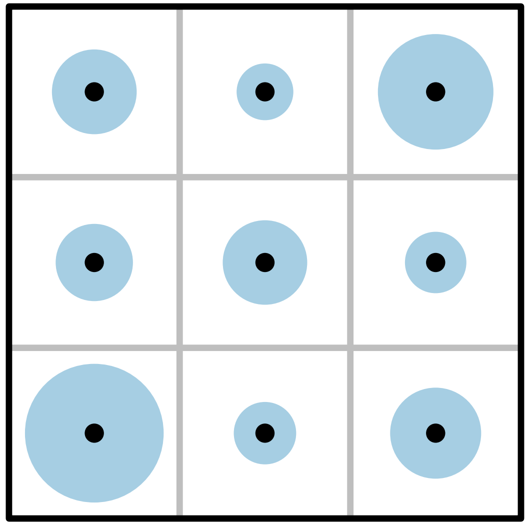
- Based on **OSM**
- $n = 50, 100, 150, 200, 250$

Quantitative Experiments



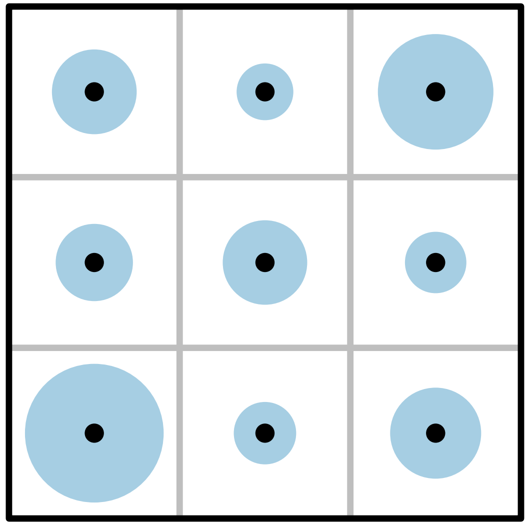
- Based on **OSM**
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Quantitative Experiments



- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Quantitative Experiments



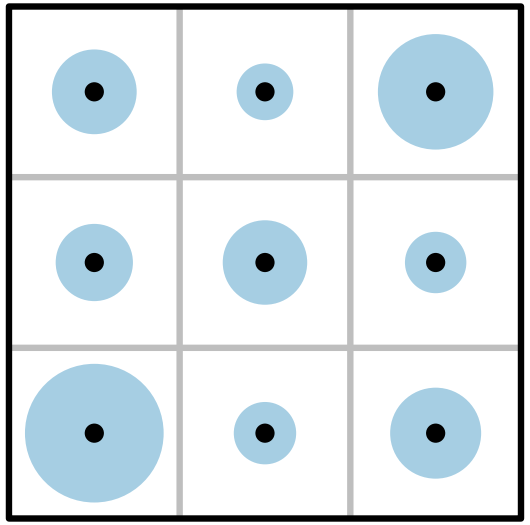
- Based on **OSM**
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

`https://github.com/JakobGeiger/ClusterSets`

Clusters

Running Time

Quantitative Experiments



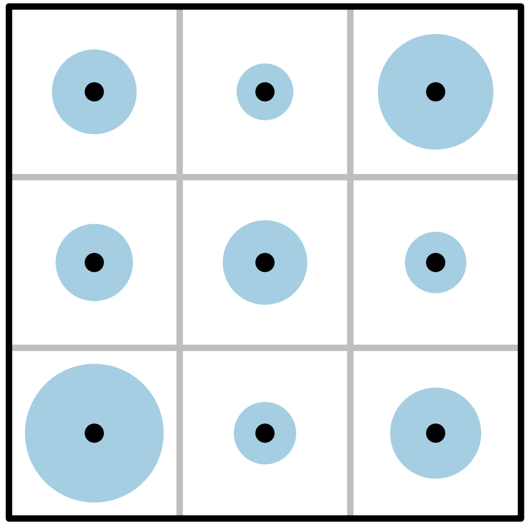
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Clusters

Running Time

GREEDY < 50 ms

Quantitative Experiments



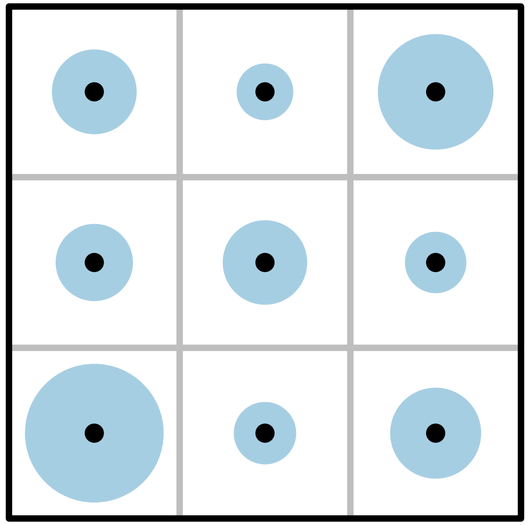
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Clusters

Running Time

GREEDY	< 50 ms
REVERSEGREEDY	< 50 ms

Quantitative Experiments



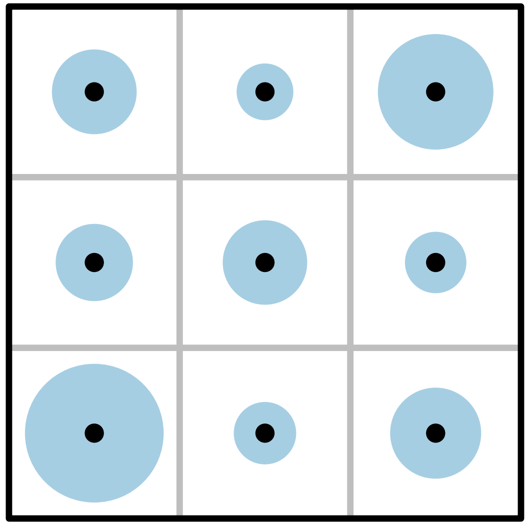
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Clusters

Running Time

Greedy	< 50 ms
ReverseGreedy	< 50 ms
ILP	

Quantitative Experiments



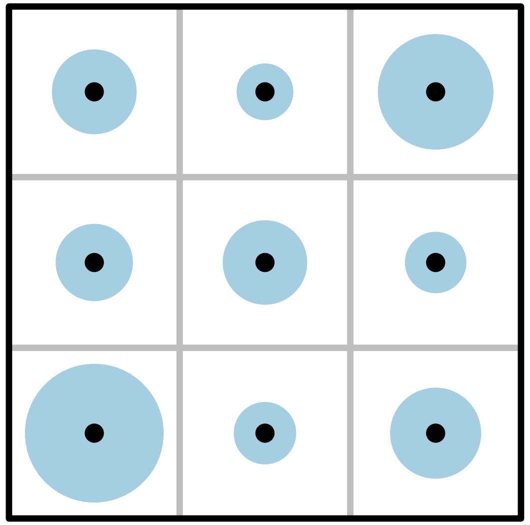
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Clusters

Running Time

Greedy		< 50 ms
ReverseGreedy		< 50 ms
ILP	50 points:	≤ 11 s

Quantitative Experiments



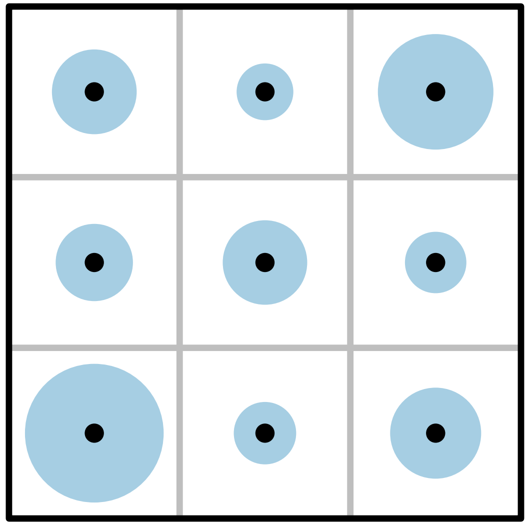
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Clusters

Running Time

GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h

Quantitative Experiments



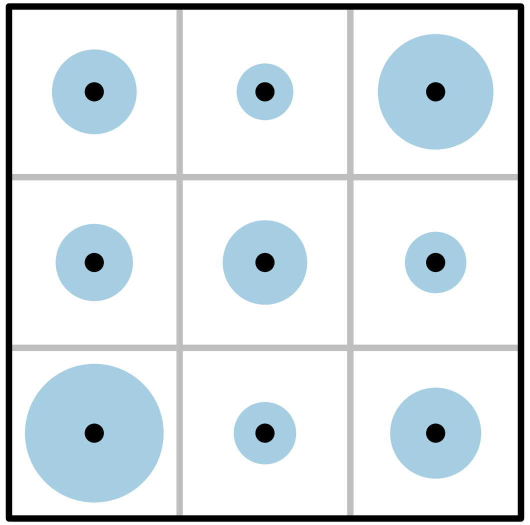
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

Clusters

Running Time

GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments



- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

$\beta = 0.5$

Clusters

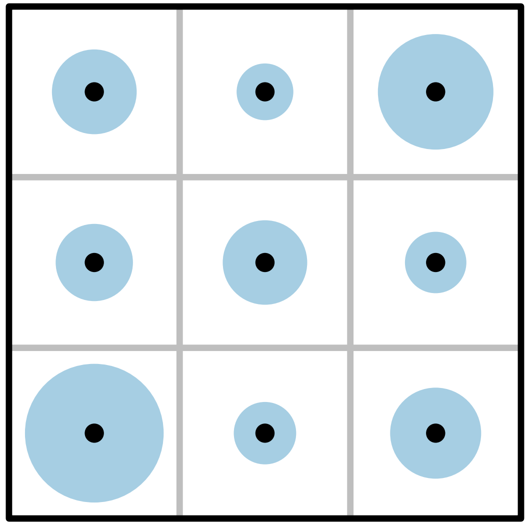
Running Time

GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments

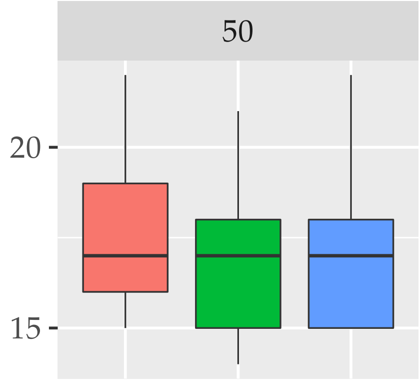
- Based on **OSM**
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

`https://github.com/JakobGeiger/ClusterSets`



$\beta = 0.5$

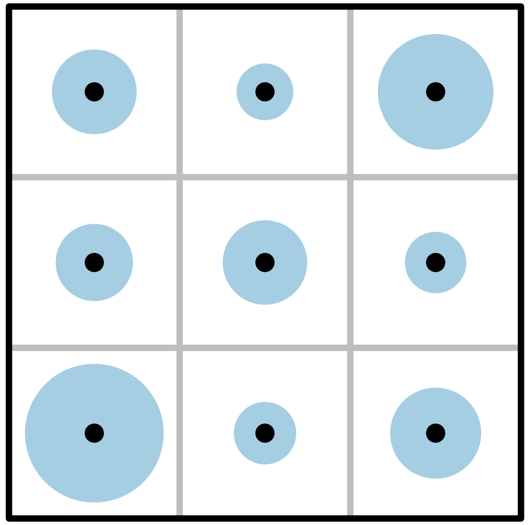
Clusters



Running Time

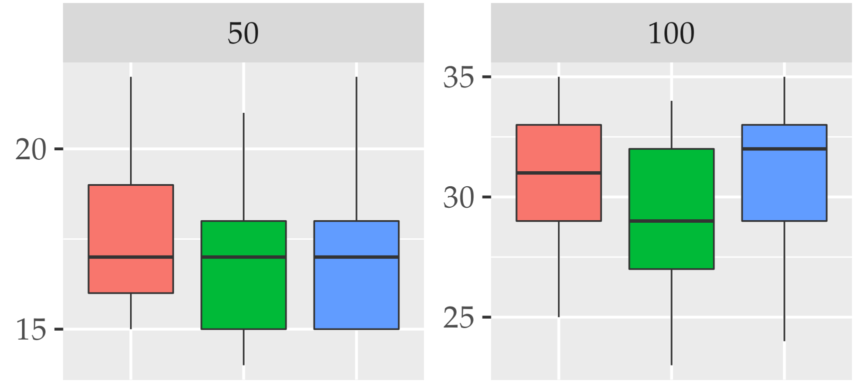
GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments



- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

$\beta = 0.5$ # Clusters

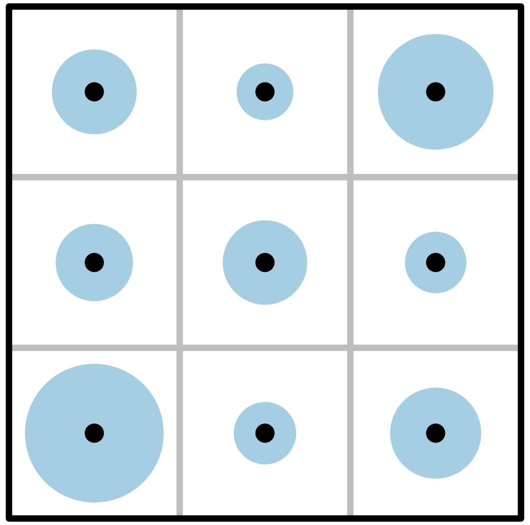


Running Time

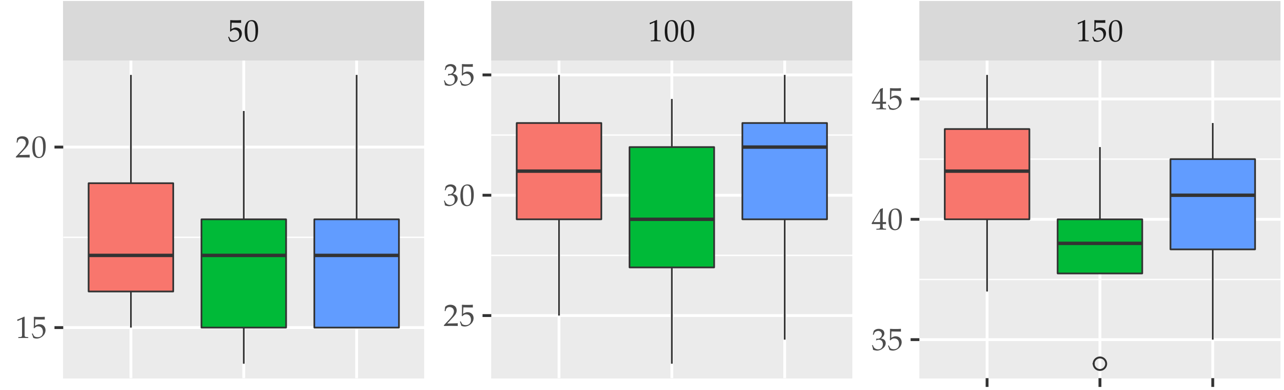
GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments

- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$



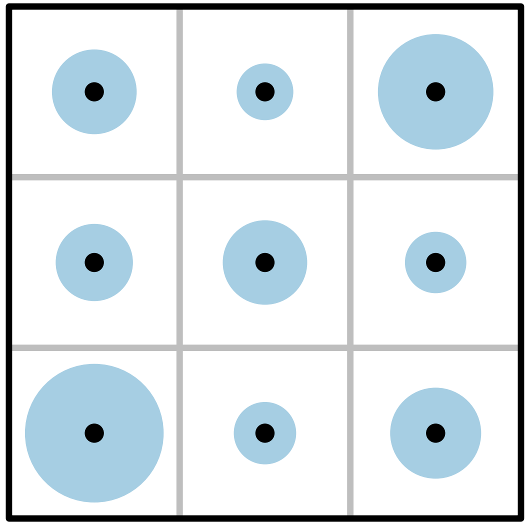
$\beta = 0.5$ # Clusters



Running Time

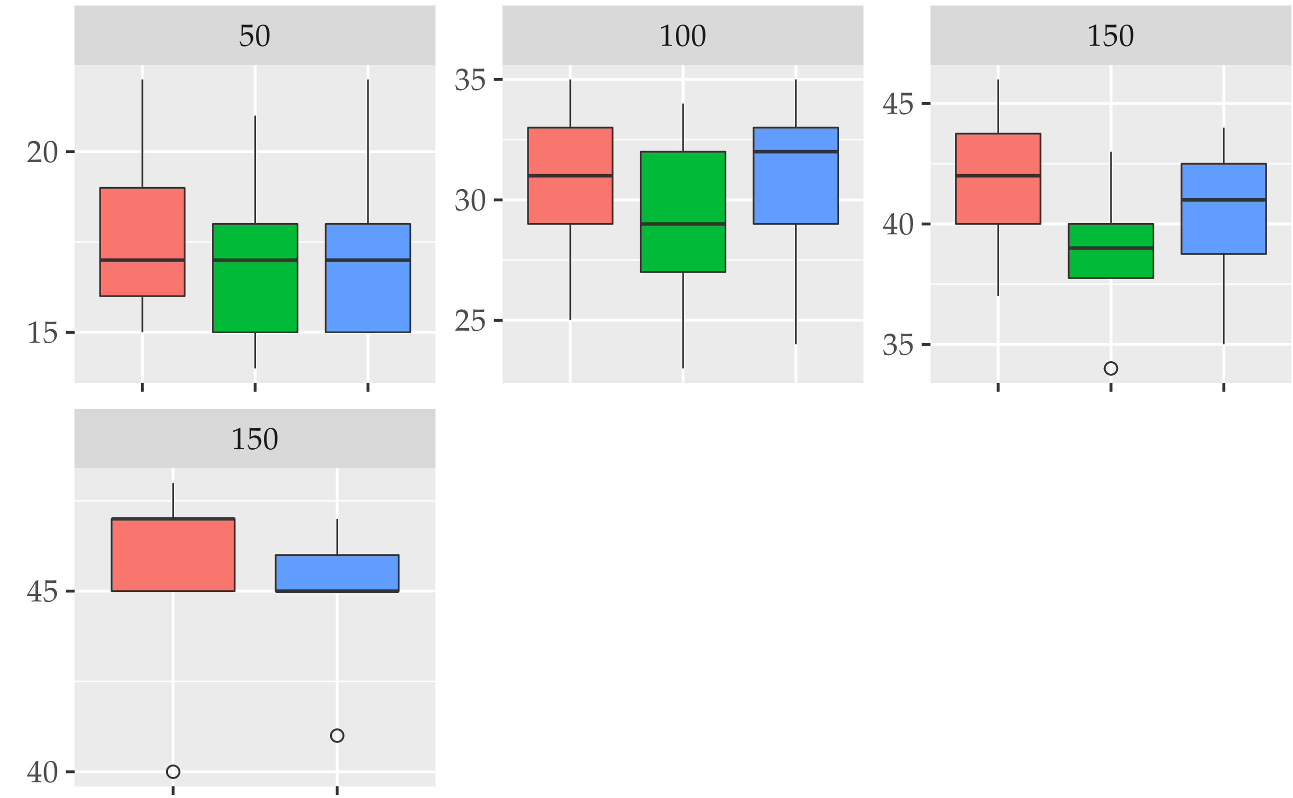
GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments



- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

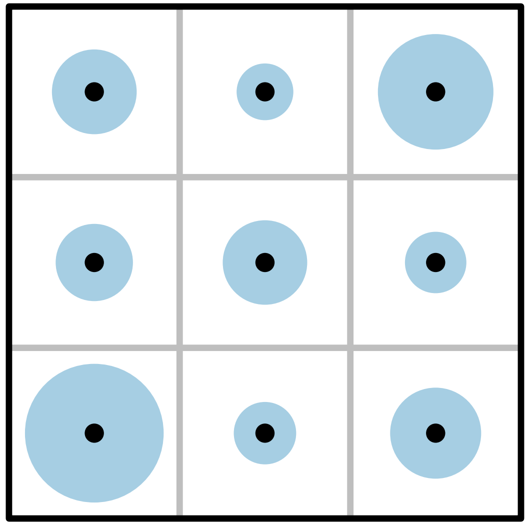
$\beta = 0.5$ # Clusters



Running Time

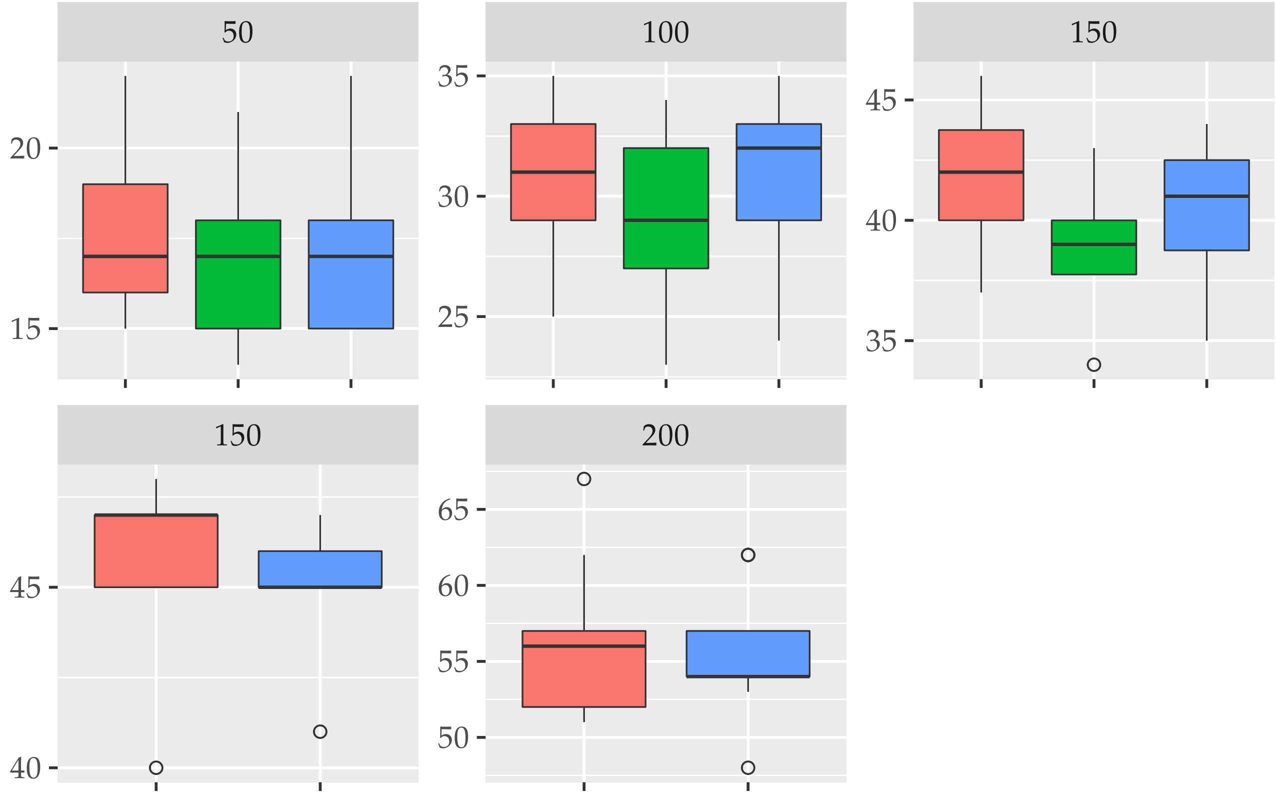
GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments



- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

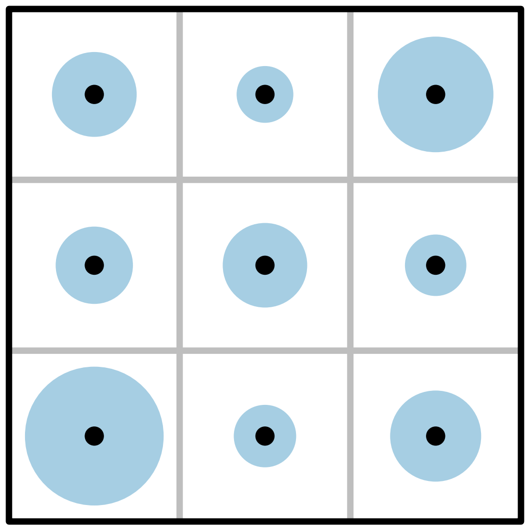
$\beta = 0.5$ # Clusters



Running Time

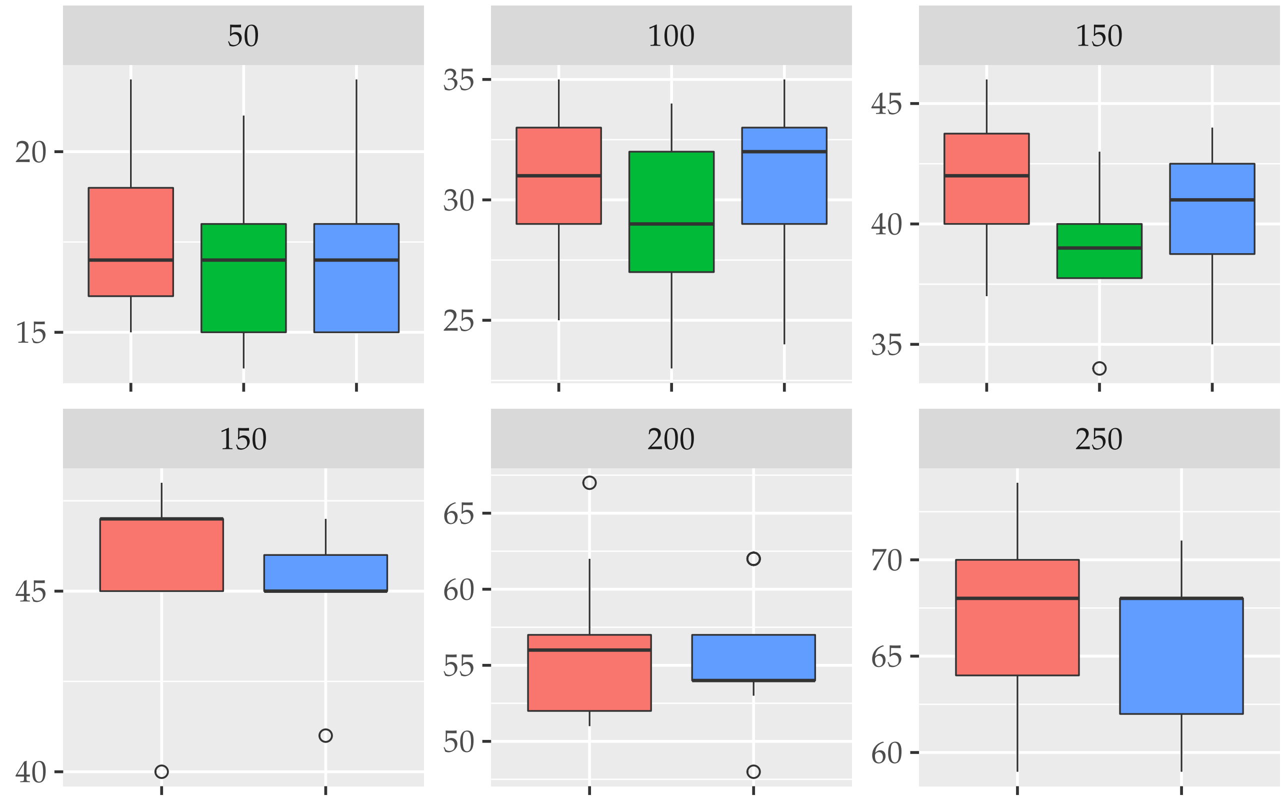
GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments



- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \dots,\ 0.9$

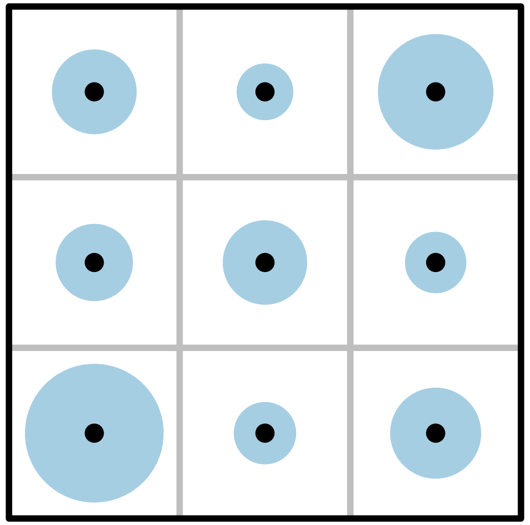
$\beta = 0.5$ # Clusters



Running Time

GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments

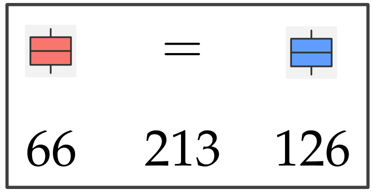


- Based on **OSM**

`https://github.com/JakobGeiger/ClusterSets`

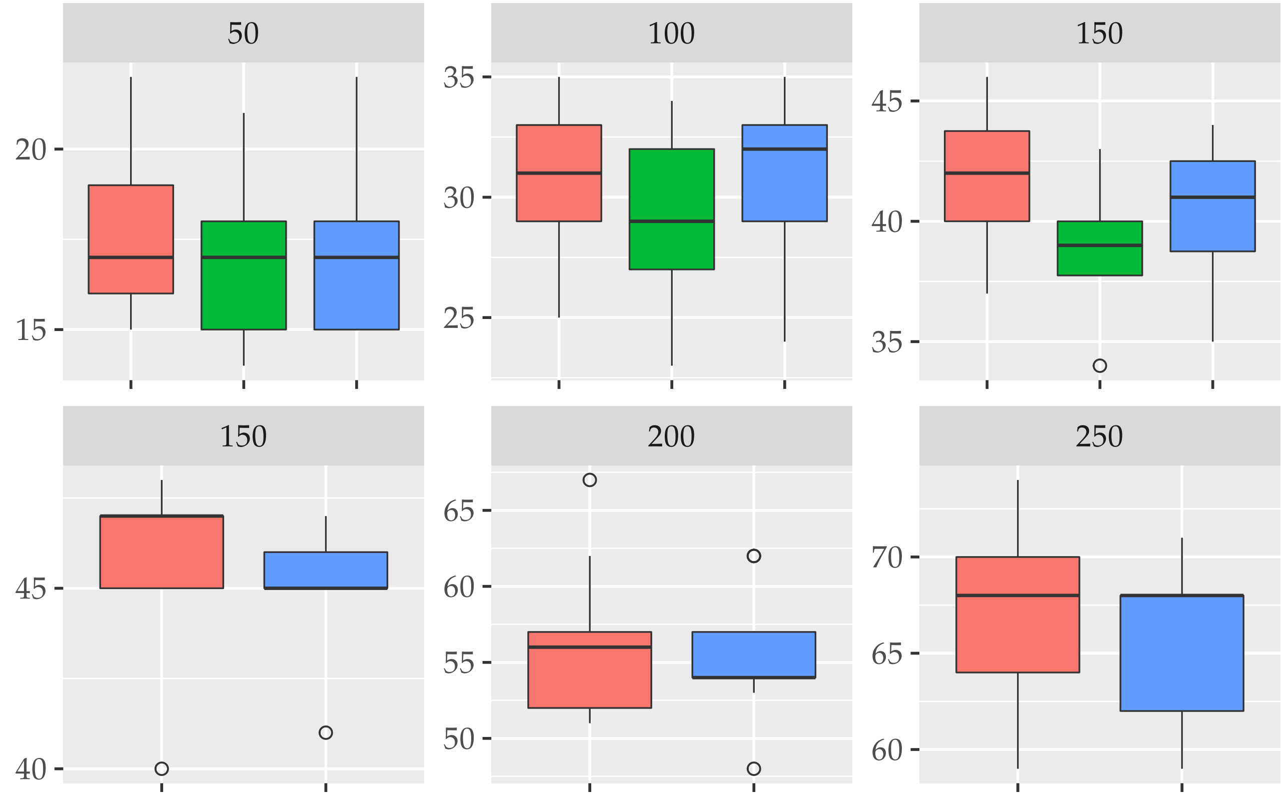
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

better:



$\beta = 0.5$

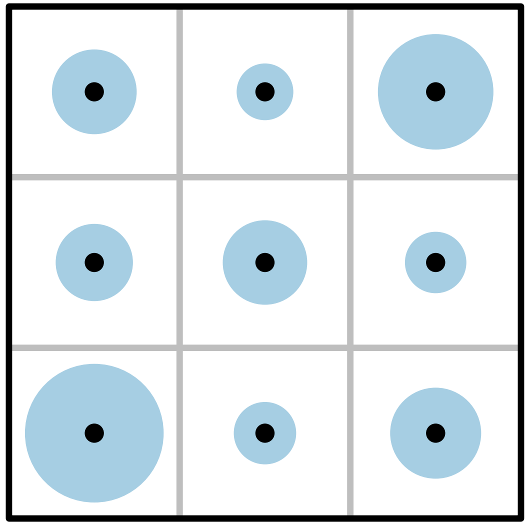
Clusters



Running Time

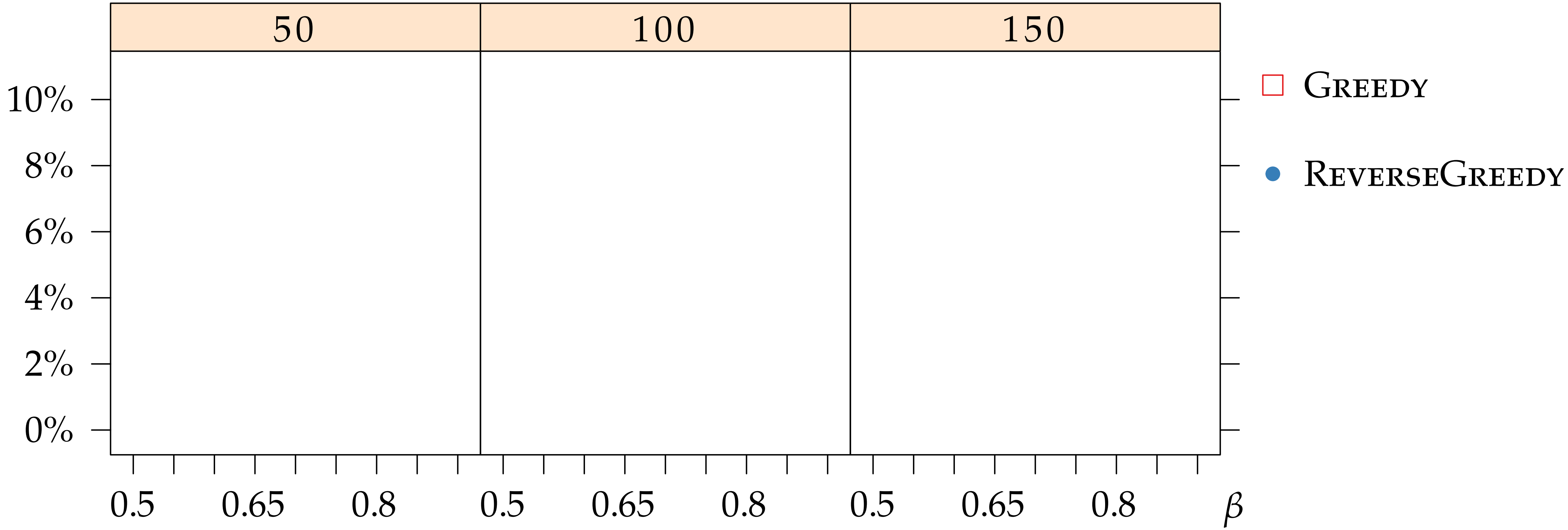
GREEDY		< 50 ms
REVERSEGREEDY		< 50 ms
ILP	50 points:	≤ 11 s
	100 points:	≤ 1.8 h
	150 points:	> 2 d

Quantitative Experiments



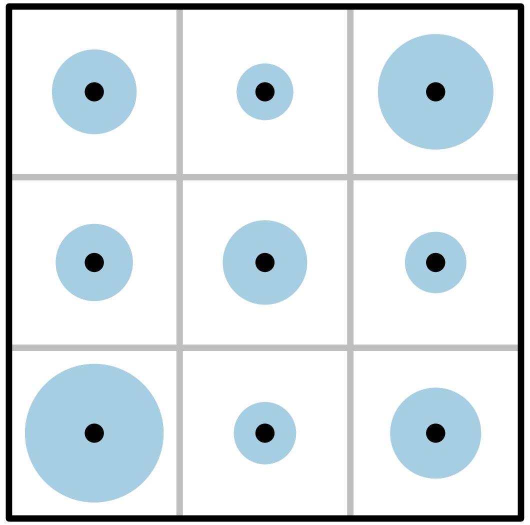
- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

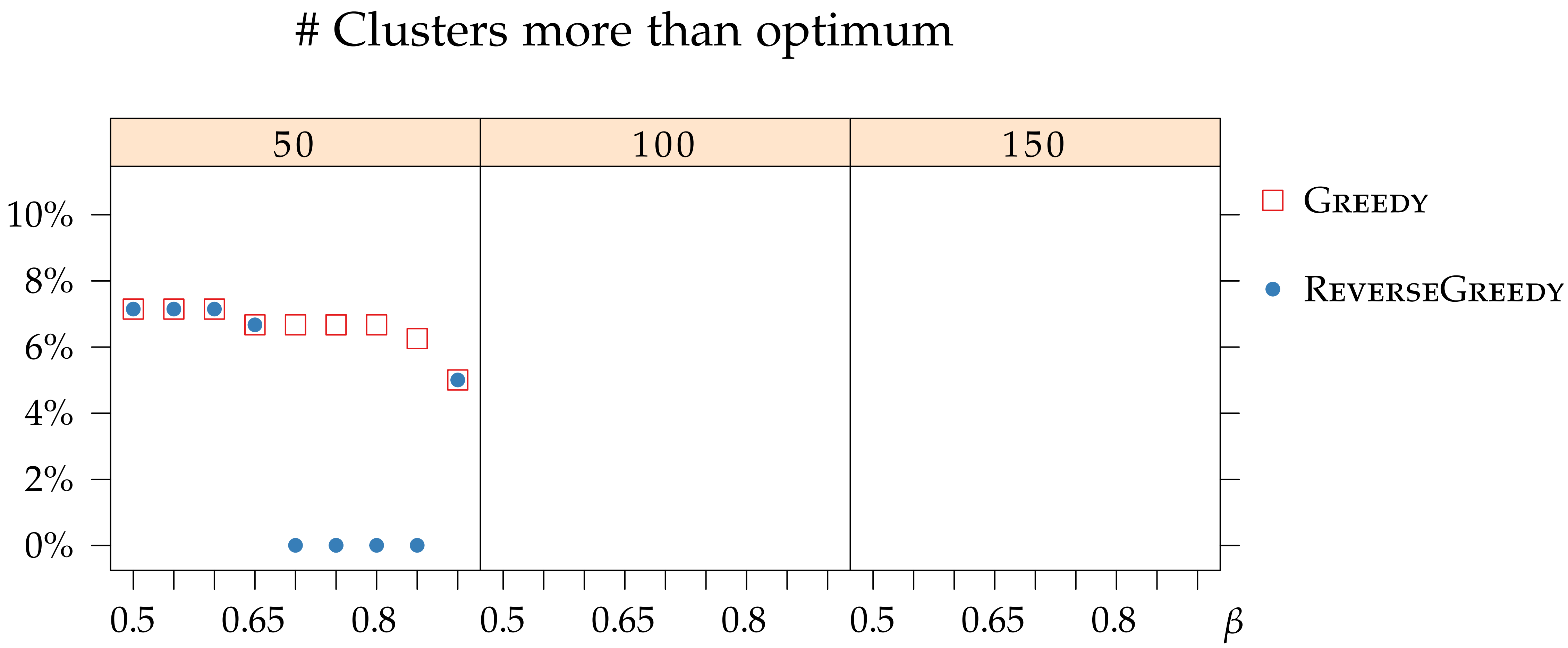
\# Clusters more than optimum



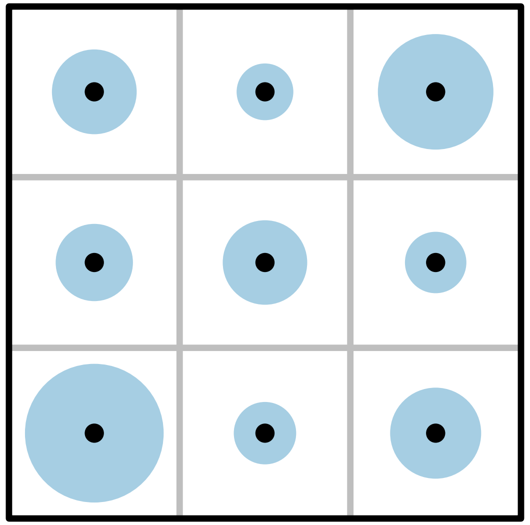
Quantitative Experiments

- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$



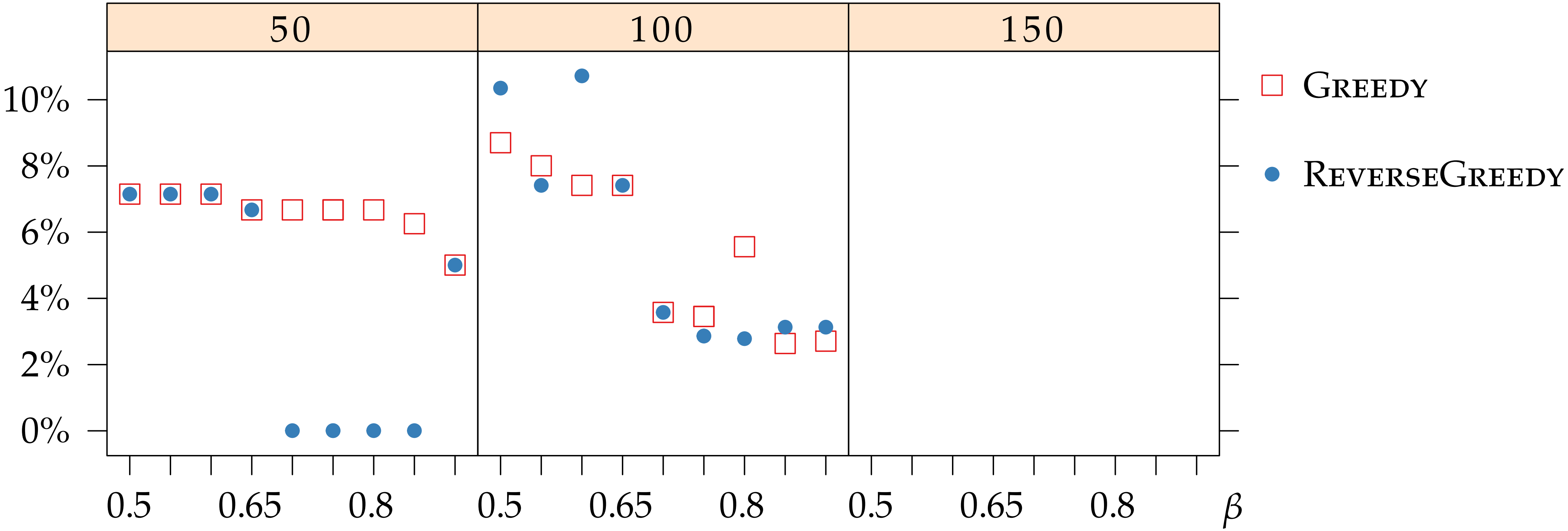


Quantitative Experiments

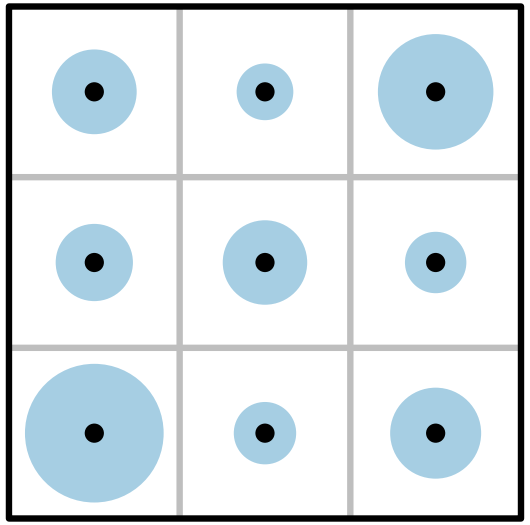


- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

\# Clusters more than optimum

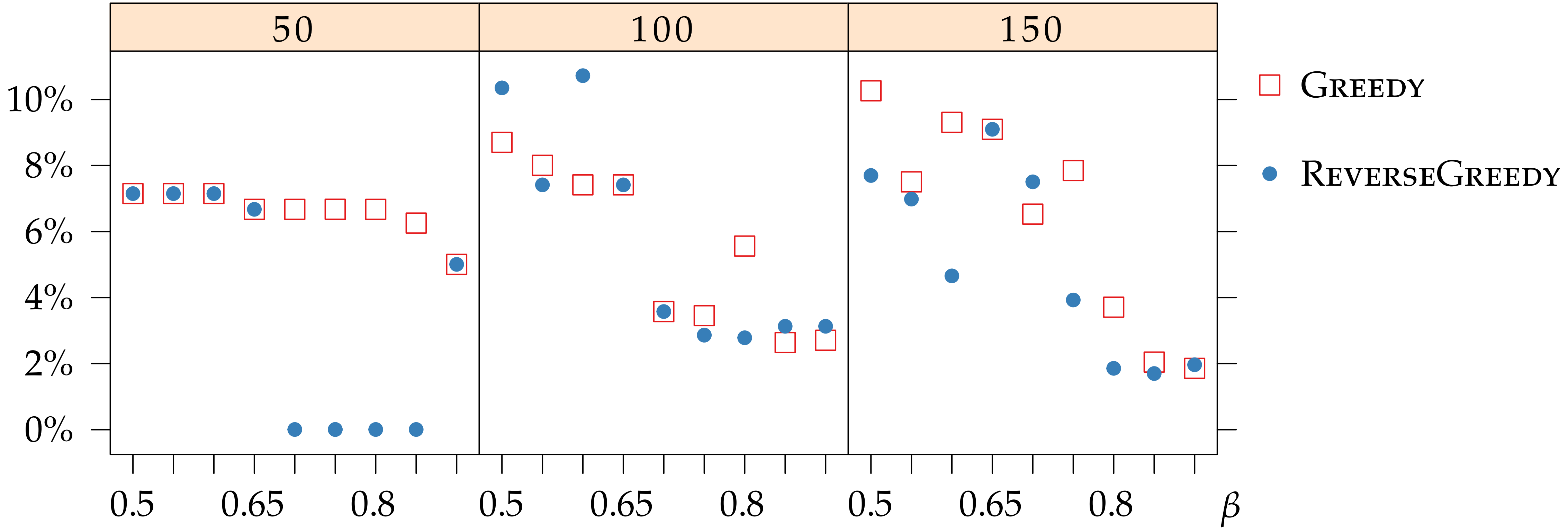


Quantitative Experiments

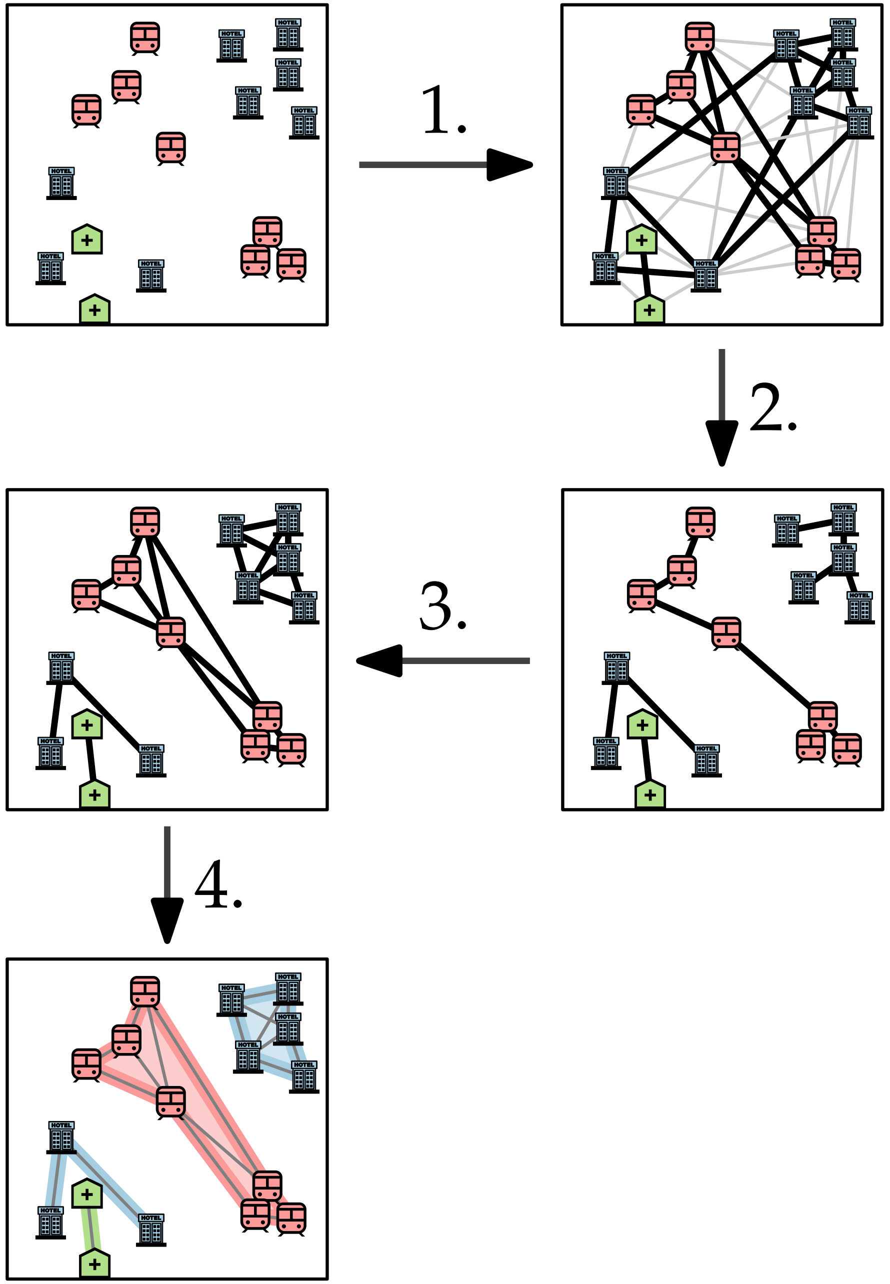


- Based on **OSM** `https://github.com/JakobGeiger/ClusterSets`
- $n = 50, 100, 150, 200, 250$
- $\beta = 0.5,\ 0.55,\ \ldots,\ 0.9$

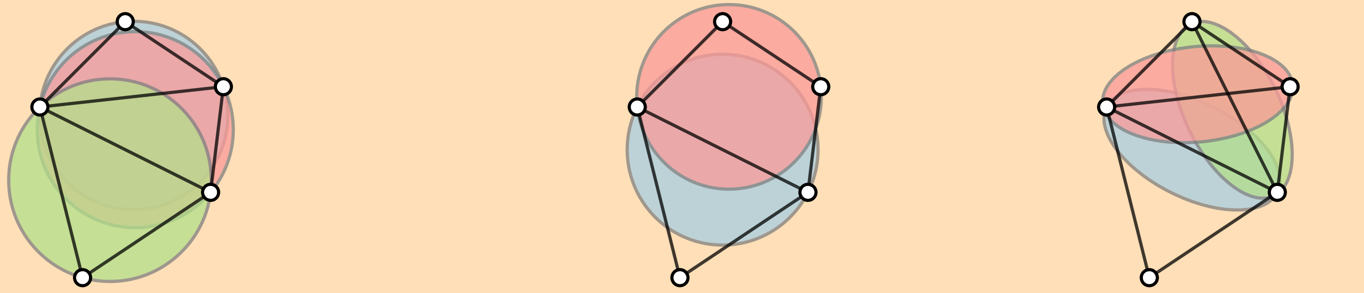
Clusters more than optimum



Conclusion



1. **Proximity Graph**

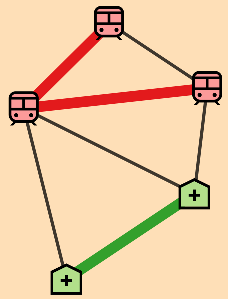


Delaunay Triangulation

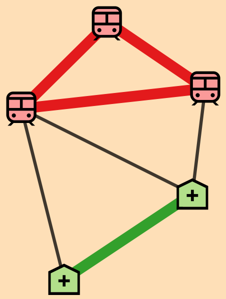
Gabriel Graph

β-Skeleton

2. **Planar Spanning Forest**



3. **Edge Augmentation**



4. **Rendering**
   - Line Voronoi Diagram
   - Tree Representation
   - Polygon Representation