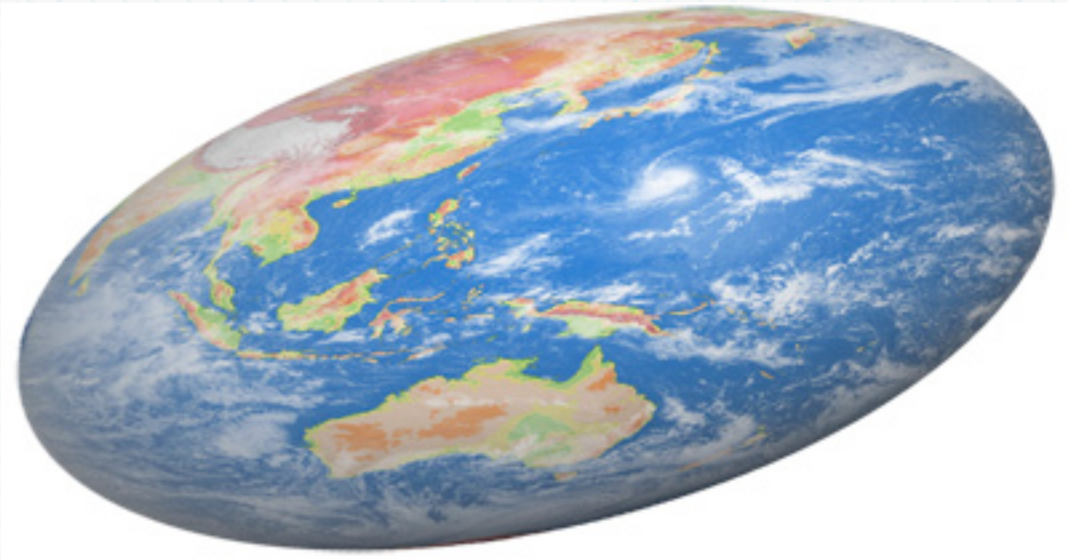
Approximation Schemes for Optimization Problems in Planar Graphs

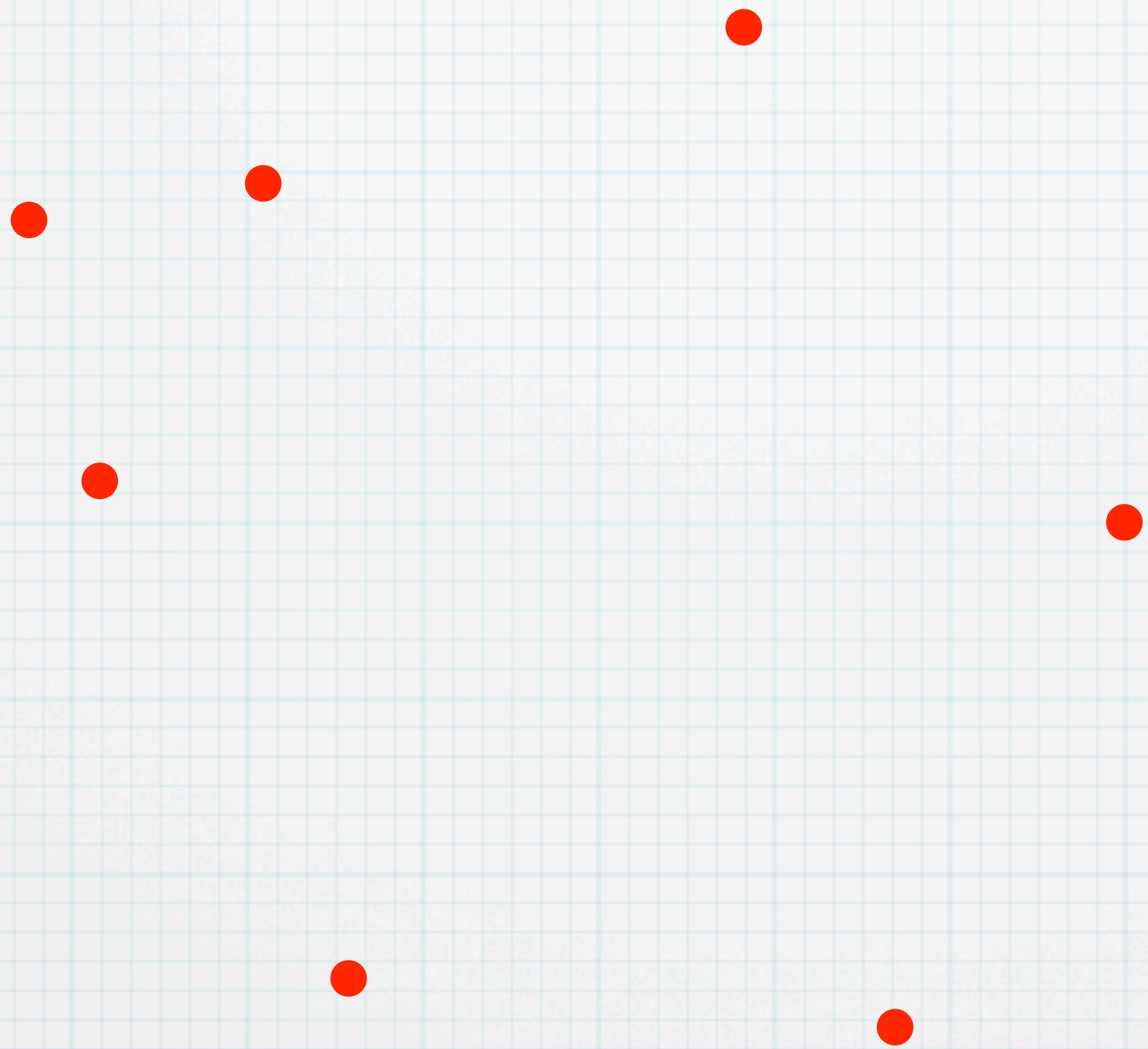
Philip Klein

BROWN

The world is flat....

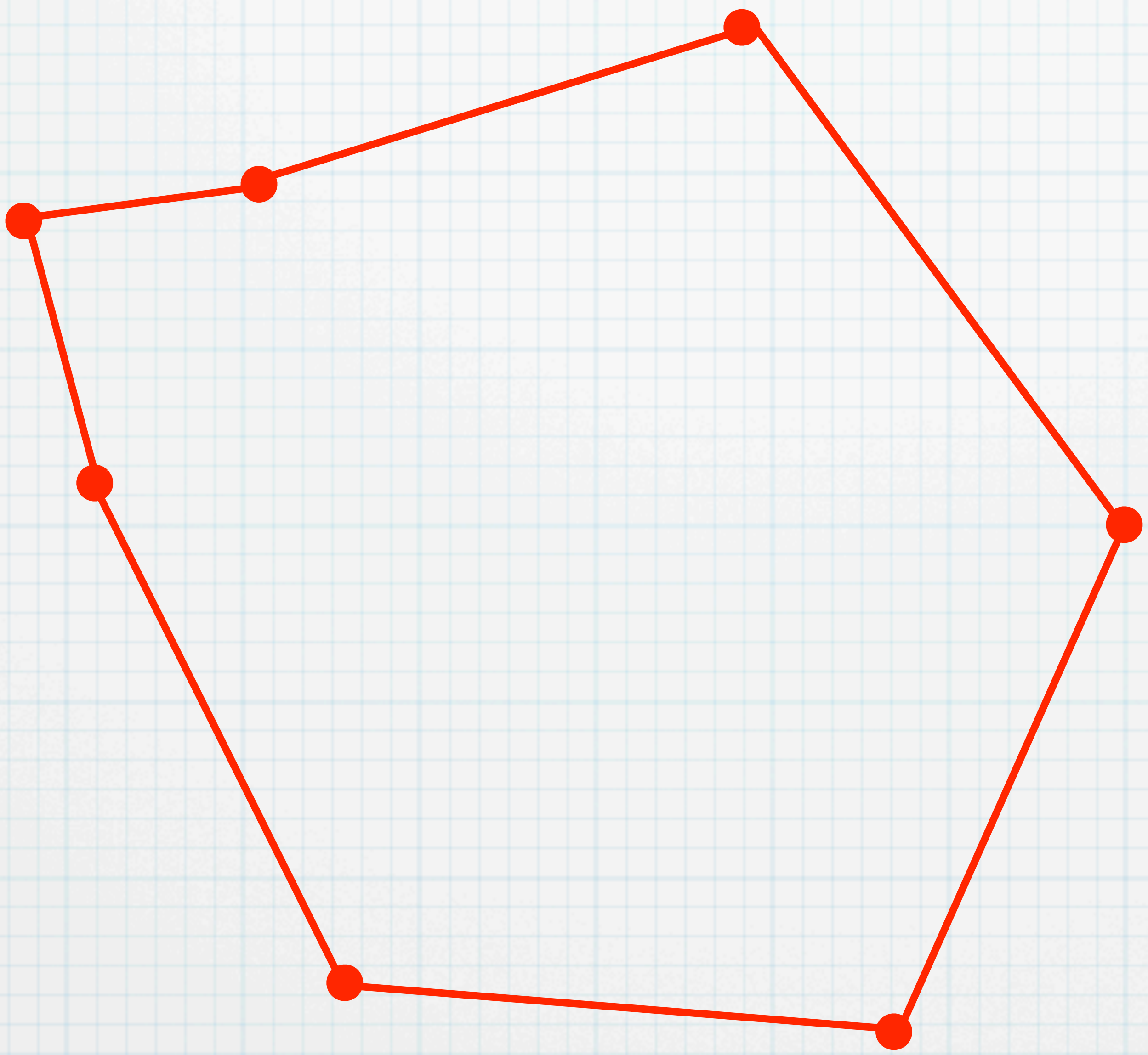


... but it's not Euclidean!



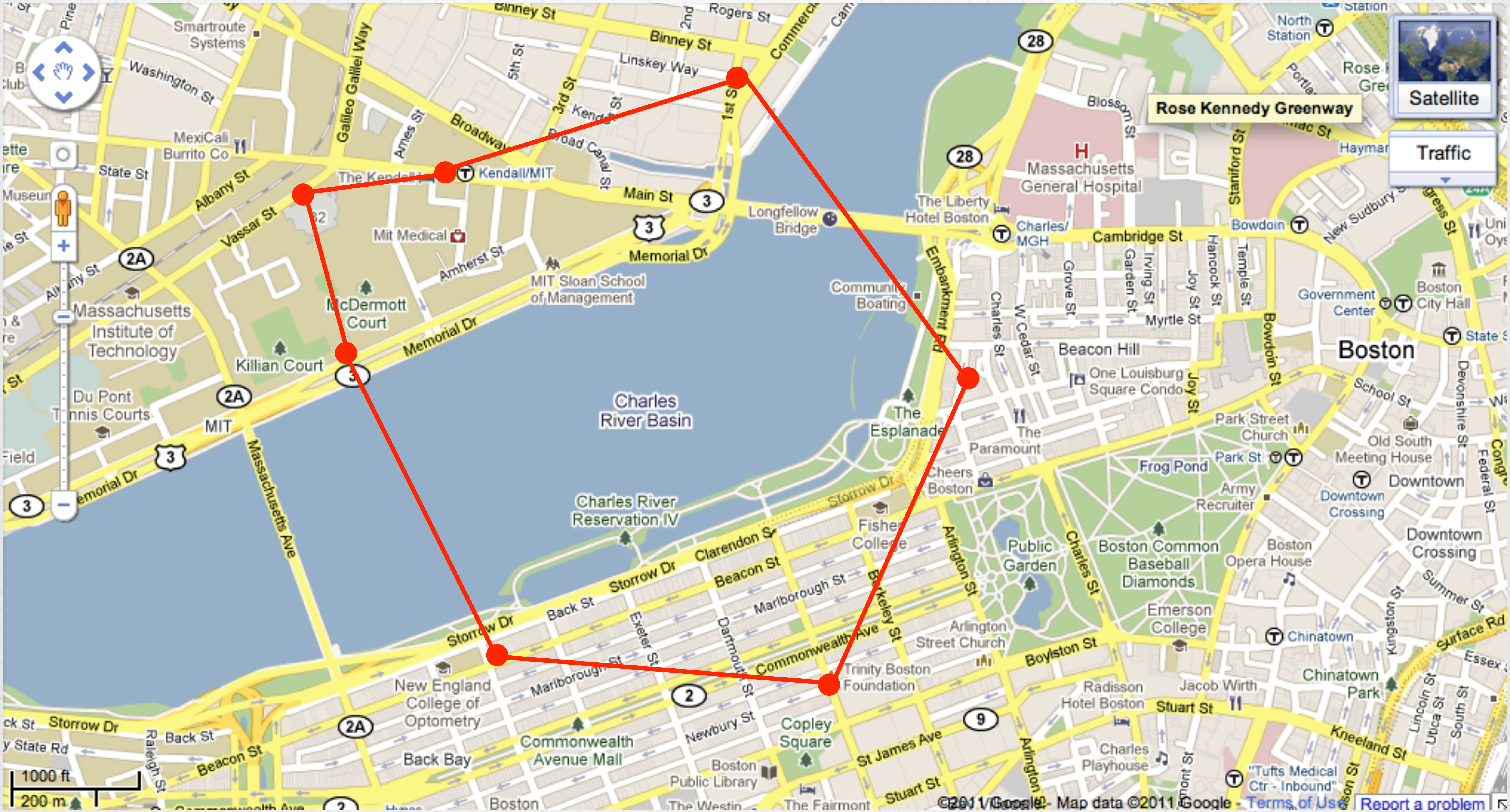
Traveling-salesman tour in the plane

... but it’s not Euclidean!



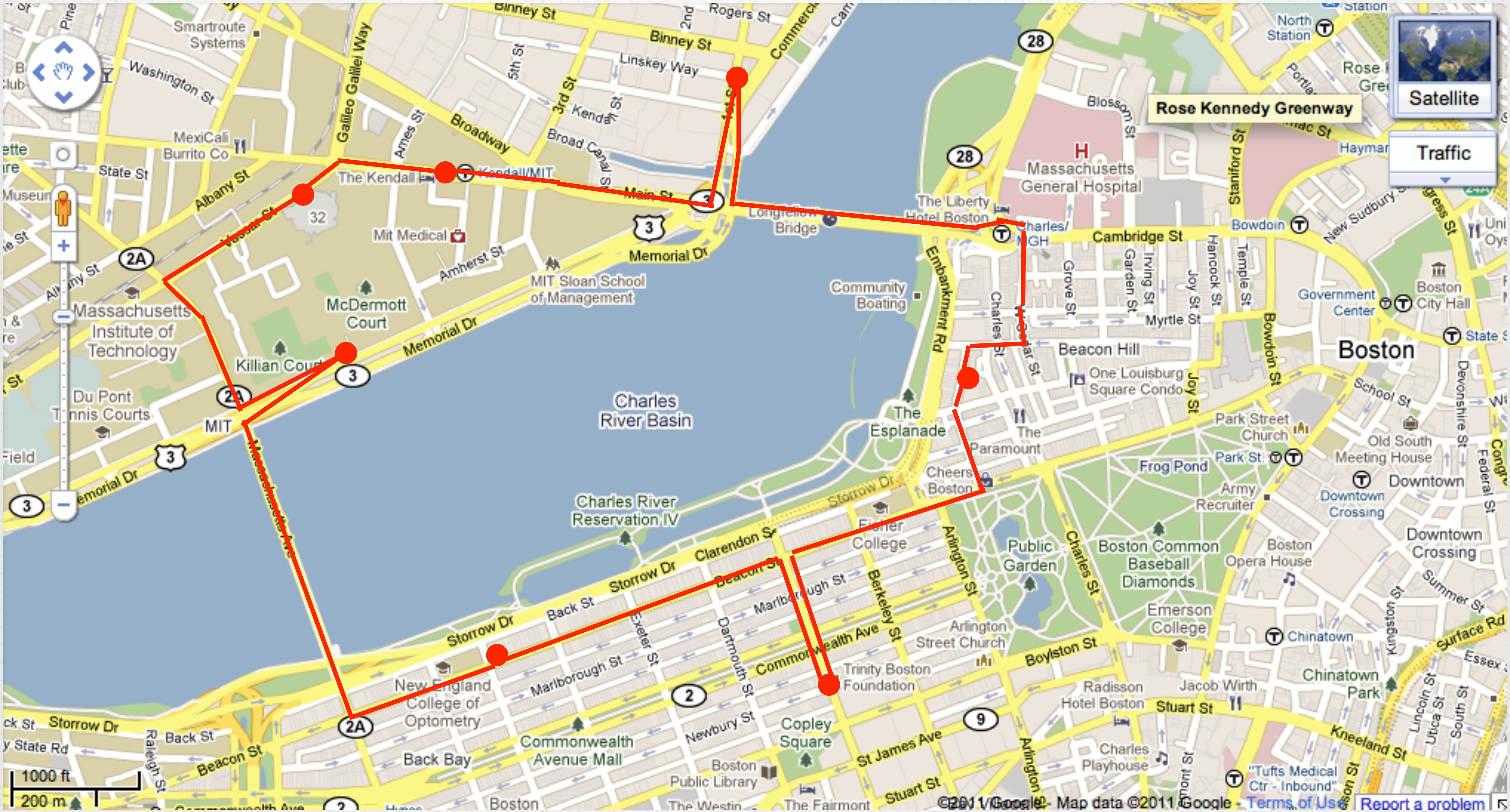
Traveling-salesman tour in the plane

... but it's not Euclidean!



Traveling-salesman tour in ~~the plane~~ a planar embedded graph

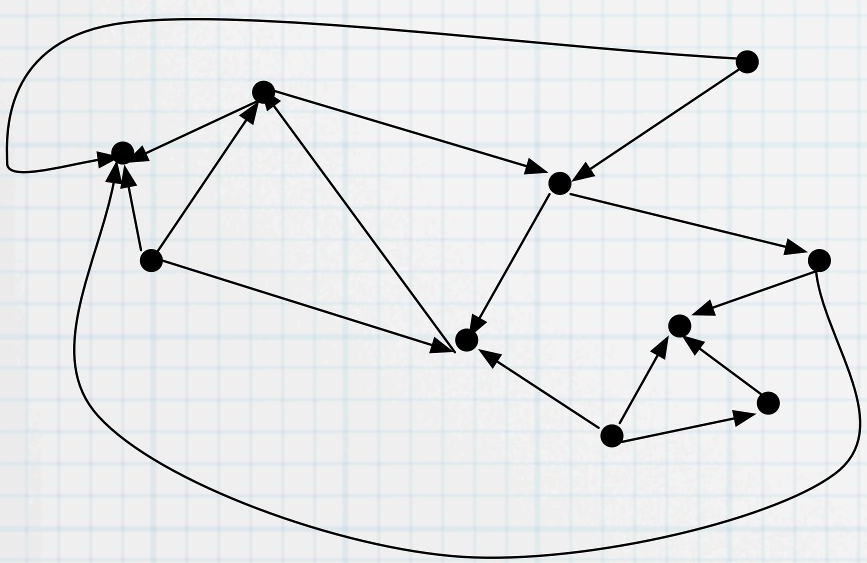
... but it's not Euclidean!

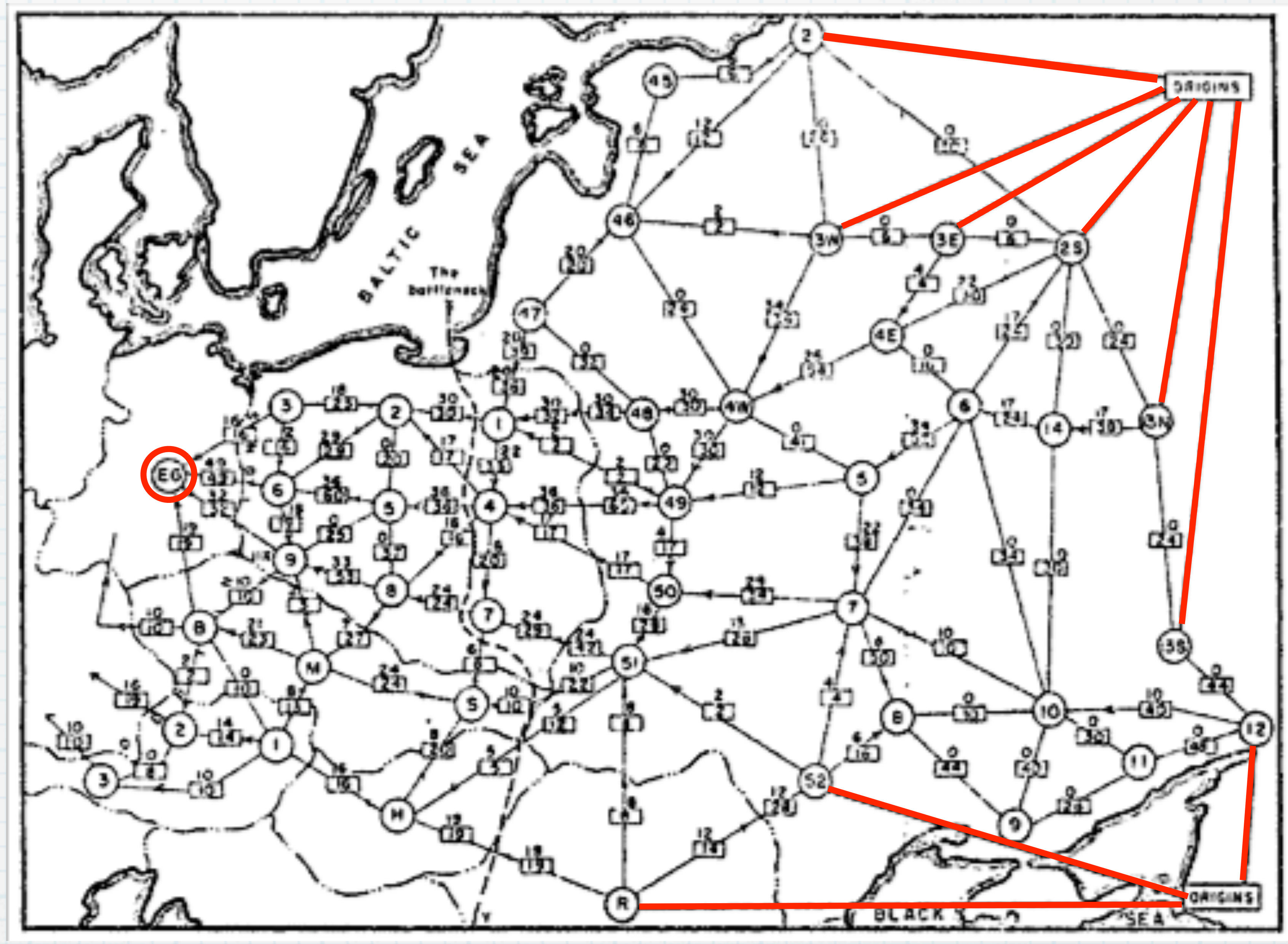


Traveling-salesman tour in ~~the plane~~ a planar embedded graph

Planar graphs

Can be drawn in the plane with no crossings

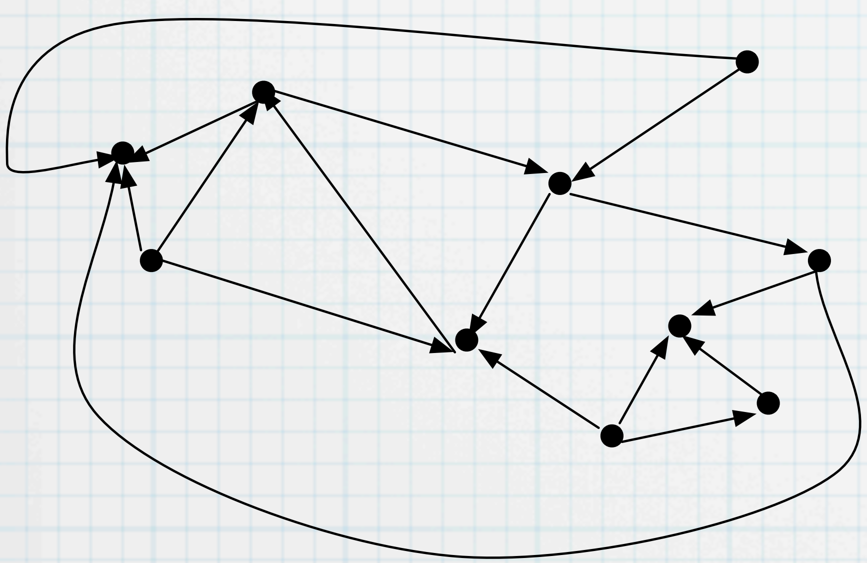


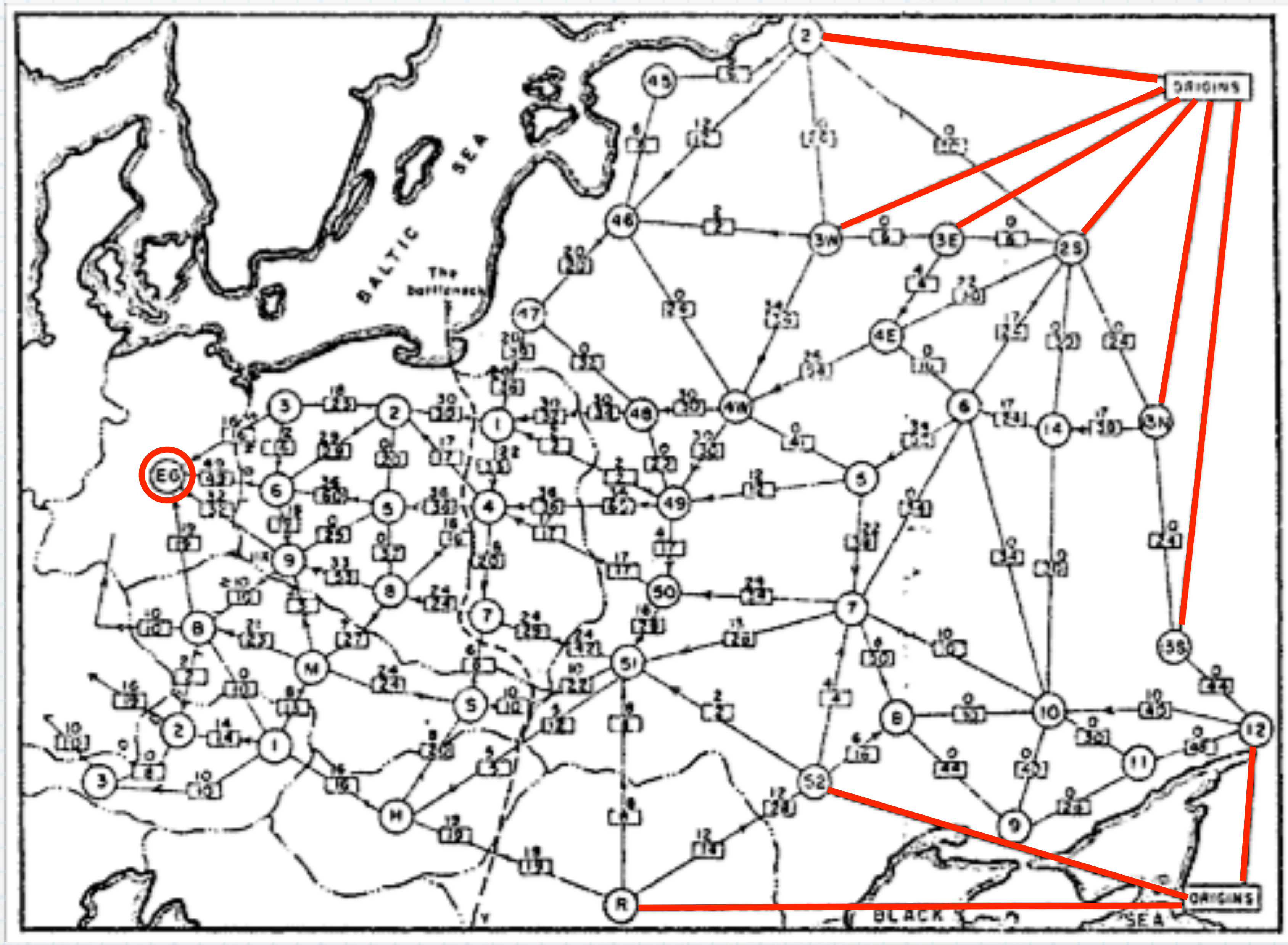


[Harris and Ross, The RAND Corporation, 1955, declassified 1999]

Planar graphs

Can be drawn in the plane with no crossings





[Harris and Ross, The RAND Corporation, 1955, declassified 1999]

Research Goal:

Exploiting planarity to achieve

- *faster* algorithms
- *more accurate* approximations

Research Goal:
Exploiting planarity to achieve
- *faster* algorithms
- *more accurate* approximations

Faster algorithms
- Shortest paths
- Maximum flow

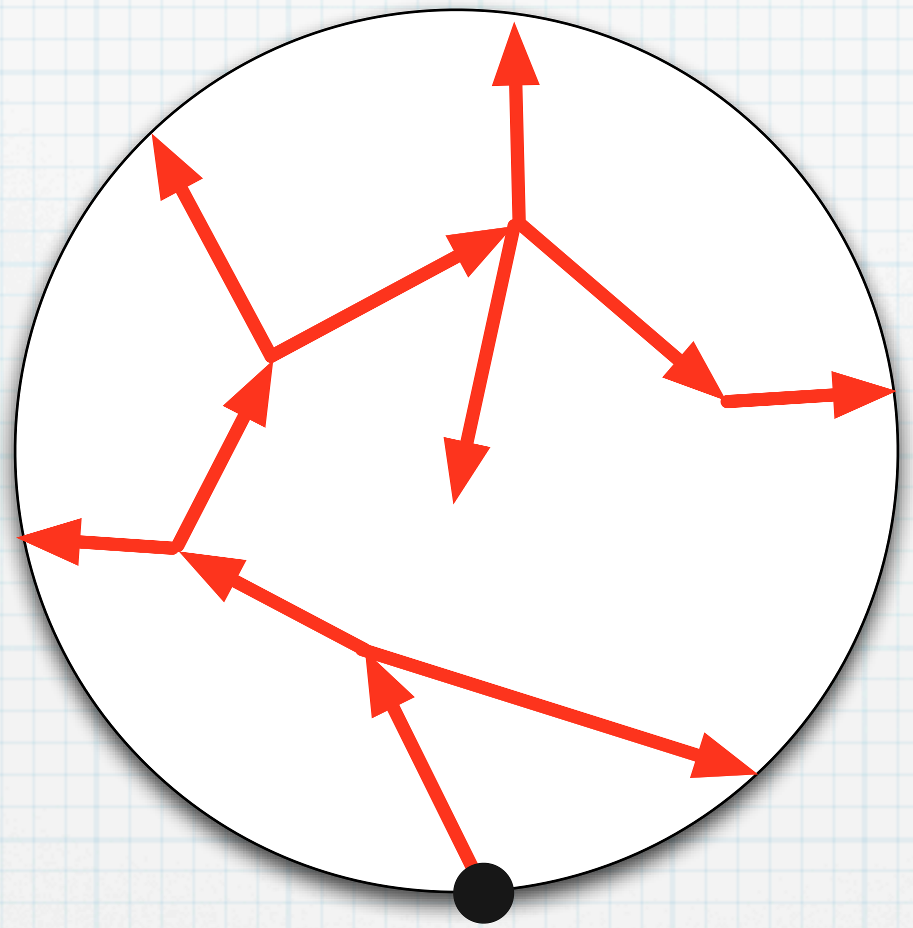
⋮

More accurate approximations
- Traveling salesman
- Steiner tree
- Multiterminal cut

⋮

Combining the two thrusts, get fast and accurate approximation algorithms.

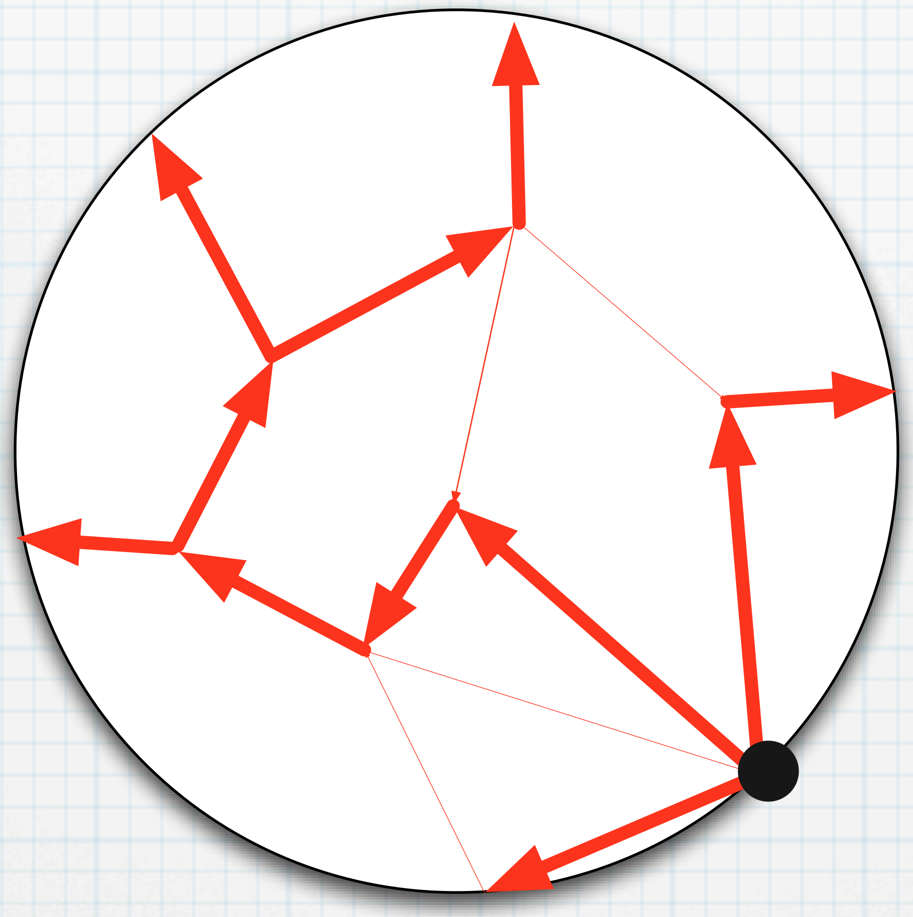
Example of faster algorithm:
Multiple-source shortest paths (MSSP)



Computes shortest-path tree rooted at each boundary node in turn.

Total time required: $O(n \log n)$

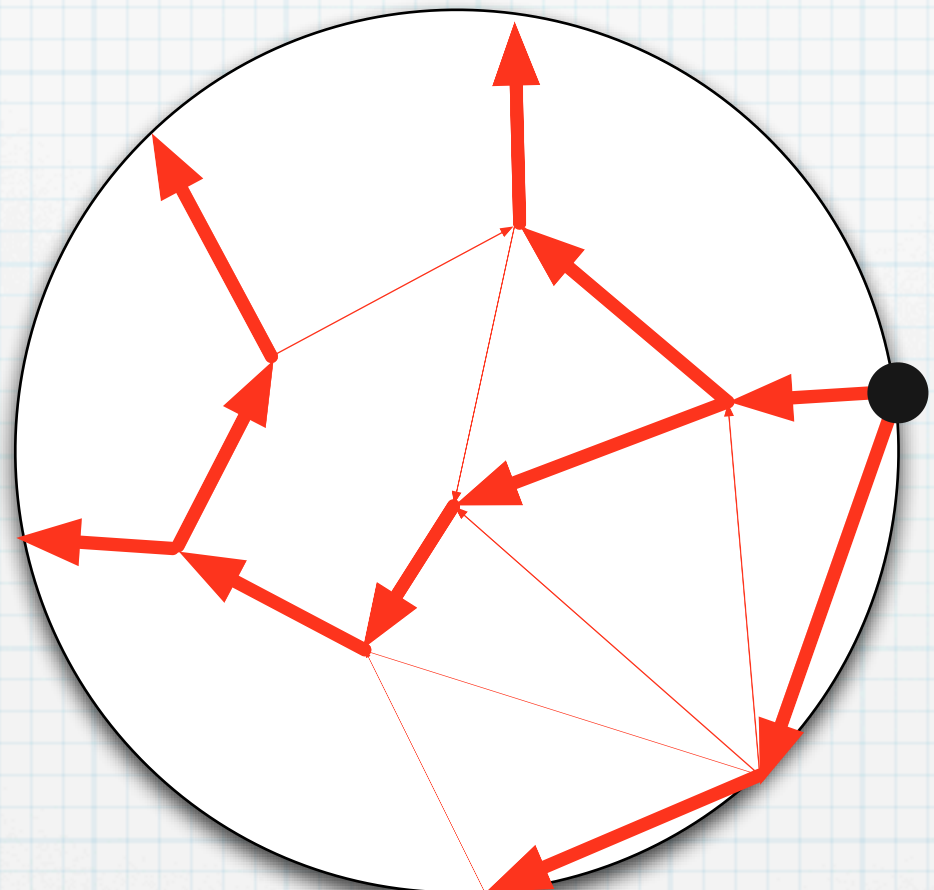
Example of faster algorithm:
Multiple-source shortest paths (MSSP)



Computes shortest-path tree rooted at each boundary node in turn.

Total time required: *O(n log n)*

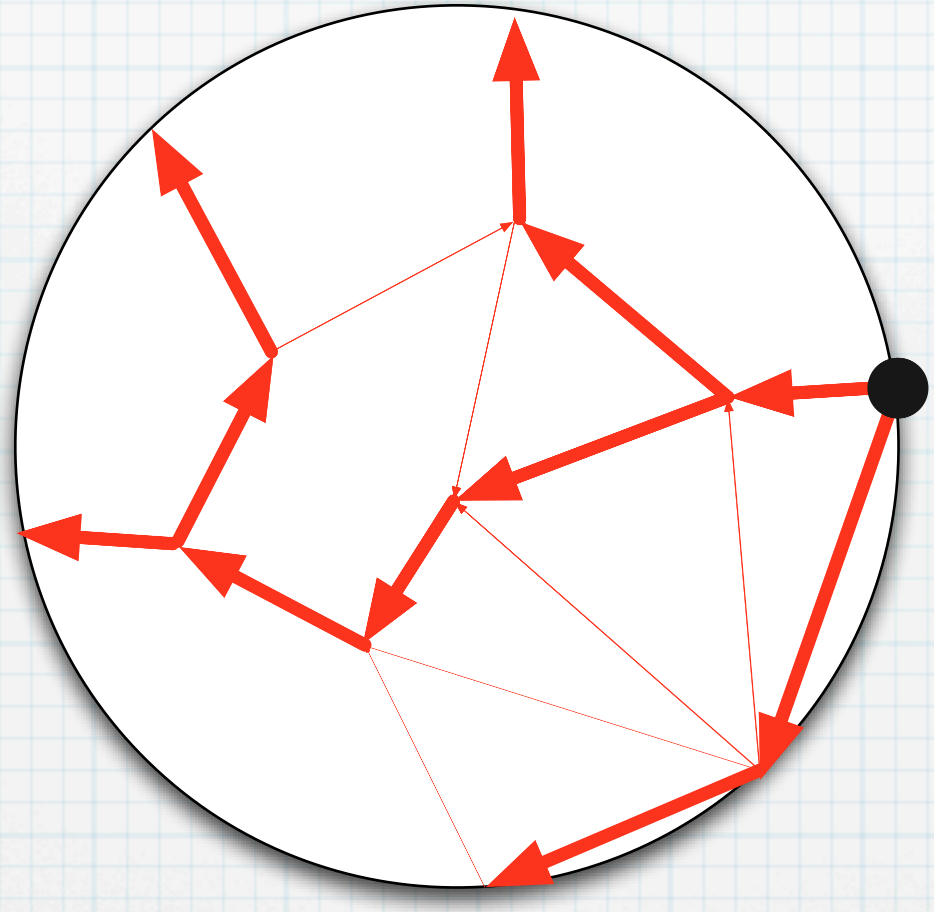
Example of faster algorithm: *Multiple-source shortest paths (MSSP)*



Computes shortest-path tree rooted at each boundary node in turn.

Total time required: $O(n \log n)$

Example of faster algorithm:
Multiple-source shortest paths (MSSP)



Computes shortest-path tree rooted at each boundary node in turn.

Total time required: $O(n \log n)$

This algorithm has turned out to have many uses----including the approximation algorithms we will discuss.

Approximation schemes for NP-hard optimization problems in planar graphs:
Greatest hits of the 70's, 80's, and 90's

1977	Lipton, Tarjan	maximum independent set	$O(n \log n)$
1983	Baker	max independent set, partition into triangles, min vertex-cover, min dominating set....	$O(n)$
1995	Grigni, Koutsoupias, Papadimitriou	Traveling salesman in unweighted graphs	$n^{O(1/\varepsilon)}$
1998	Arora, Grigni, Karger, Klein, Woloszyn	Traveling salesman in graphs with weights	$n^{O(1/\varepsilon^2)}$

Approximation schemes for NP-hard optimization problems in planar graphs:

Greatest hits of the 70's, 80's, and 90's

1977	Lipton, Tarjan	maximum independent set	$O(n \log n)$
1983	Baker	max independent set, partition into triangles, min vertex-cover, min dominating set....	$O(n)$
1995	Grigni, Koutsoupias, Papadimitriou	Traveling salesman in unweighted graphs	$\cancel{n^{O(1/\varepsilon)}}$
1998	Arora, Grigni, Karger, Klein, Woloszyn	Traveling salesman in graphs with weights	$\cancel{n^{O(1/\varepsilon^2)}}$

Definition: An approximation scheme is *efficient* if running time is a polynomial whose degree is fixed independent of ε

Approximation schemes for NP-hard optimization problems in planar graphs:
Greatest hits of the 70's, 80's, and 90's

1977	Lipton, Tarjan	maximum independent set	$O(n \log n)$
1983	Baker	max independent set, partition into triangles, min vertex-cover, min dominating set....	$O(n)$
1995	Grigni, Koutsoupias, Papadimitriou	Traveling salesman in unweighted graphs	$\cancel{n^{O(1/\varepsilon)}}$
1998	Arora, Grigni, Karger, Klein, Woloszyn	Traveling salesman in graphs with weights	$\cancel{n^{O(1/\varepsilon^2)}}$

Definition: An approximation scheme is *efficient* if running time is a polynomial whose degree is fixed independent of ε

For the 00's: give *efficient* approximation scheme for TSP, address greater variety of traditional optimization problems.

Question: Is there an *efficient* approximation scheme for traveling salesman?

Theorem [Klein, 2005]: There is a linear-time approximation scheme for the traveling-salesman problem in planar graphs with weights

The framework introduced by this paper has since been used to address many other problems....

Use of new framework for approximation schemes for planar graphs

- Traveling salesman [Klein, 2005]
- Traveling salesman on subset of vertices [Klein, 2006]
- 2-edge-connected spanning subgraph
  [Berger, Grigni, 2007]
- Steiner tree [Borradaile, Klein, Mathieu, 2008]
- 2-edge-connected Steiner multisubgraph
  [Borradaile, Klein, 2008]
- Steiner forest [Bateni, Hajiaghayi, Marx, 2010]
  speed-up [Eisenstat et al., new]
- Prize-collecting Steiner tree, TSP, stroll
  [Bateni, Chekuri, Ene, Hajiaghayi, Korula, Marx, 2011]
- Multiterminal cut [Bateni, Hajiaghayi, K., Mathieu, unpublished]

Use of new framework for approximation schemes for planar graphs

- Traveling salesman [Klein, 2005]
- Traveling salesman on subset of vertices [Klein, 2006]
- 2-edge-connected spanning subgraph [Berger, Grigni, 2007]
- Steiner tree [Borradaile, Klein, Mathieu, 2008]
- 2-edge-connected Steiner multisubgraph [Borradaile, Klein, 2008]
- Steiner forest [Bateni, Hajiaghayi, Marx, 2010] speed-up [Eisenstat et al., new]
- Prize-collecting Steiner tree, TSP, stroll [Bateni, Chekuri, Ene, Hajiaghayi, Korula, Marx, 2011]
- Multiterminal cut [Bateni, Hajiaghayi, K., Mathieu, unpublished]

Framework generalized to broader graph classes

- Steiner tree in bounded-genus graphs [Borradaile, Demaine, Tazari, 2009]
- TSP in excluded-minor graphs [Demaine, Hajiaghayi, and Kawarabayashi, 2011]

Use of new framework for approximation schemes for planar graphs

Time efficient?

- Traveling salesman [Klein, 2005]
- Traveling salesman on subset of vertices [Klein, 2006]
- 2-edge-connected spanning subgraph [Berger, Grigni, 2007]
- Steiner tree [Borradaile, Klein, Mathieu, 2008]
- 2-edge-connected Steiner multisubgraph [Borradaile, Klein, 2008]
- Steiner forest [Bateni, Hajiaghayi, Marx, 2010]
  speed-up [Eisenstat et al., new]
- Prize-collecting Steiner tree, TSP, stroll [Bateni, Chekuri, Ene, Hajiaghayi, Korula, Marx, 2011]
- Multiterminal cut [Bateni, Hajiaghayi, K., Mathieu, unpublished]

Framework generalized to broader graph classes

- Steiner tree in bounded-genus graphs [Borradaile, Demaine, Tazari, 2009]
- TSP in excluded-minor graphs [Demaine, Hajiaghayi, and Kawarabayashi, 2011]

Use of new framework for approximation schemes for planar graphs

	Time efficient?
• Traveling salesman [Klein, 2005]	$O(n)$
• Traveling salesman on subset of vertices [Klein, 2006]	$O(n \log n)$
• 2-edge-connected spanning subgraph [Berger, Grigni, 2007]	unit-weights: $O(n)$ general weights: $O(n^{f(\varepsilon)})$
• Steiner tree [Borradaile, Klein, Mathieu, 2008]	$O(n \log n)$
• 2-edge-connected Steiner multisubgraph [Borradaile, Klein, 2008]	$O(n \log n)$
• Steiner forest [Bateni, Hajiaghayi, Marx, 2010]	$O(n^{f(\varepsilon)})$
speed-up [Eisenstat et al., new]	$O(n \log^{f(\varepsilon)} n)$
• Prize-collecting Steiner tree, TSP, stroll [Bateni, Chekuri, Ene, Hajiaghayi, Korula, Marx, 2011]	$O(n^c)$
• Multiterminal cut [Bateni, Hajiaghayi, K., Mathieu, unpublished]	$O(n^c)$

Framework generalized to broader graph classes

- Steiner tree in bounded-genus graphs [Borradaile, Demaine, Tazari, 2009]
- TSP in excluded-minor graphs [Demaine, Hajiaghayi, and Kawarabayashi, 2011]

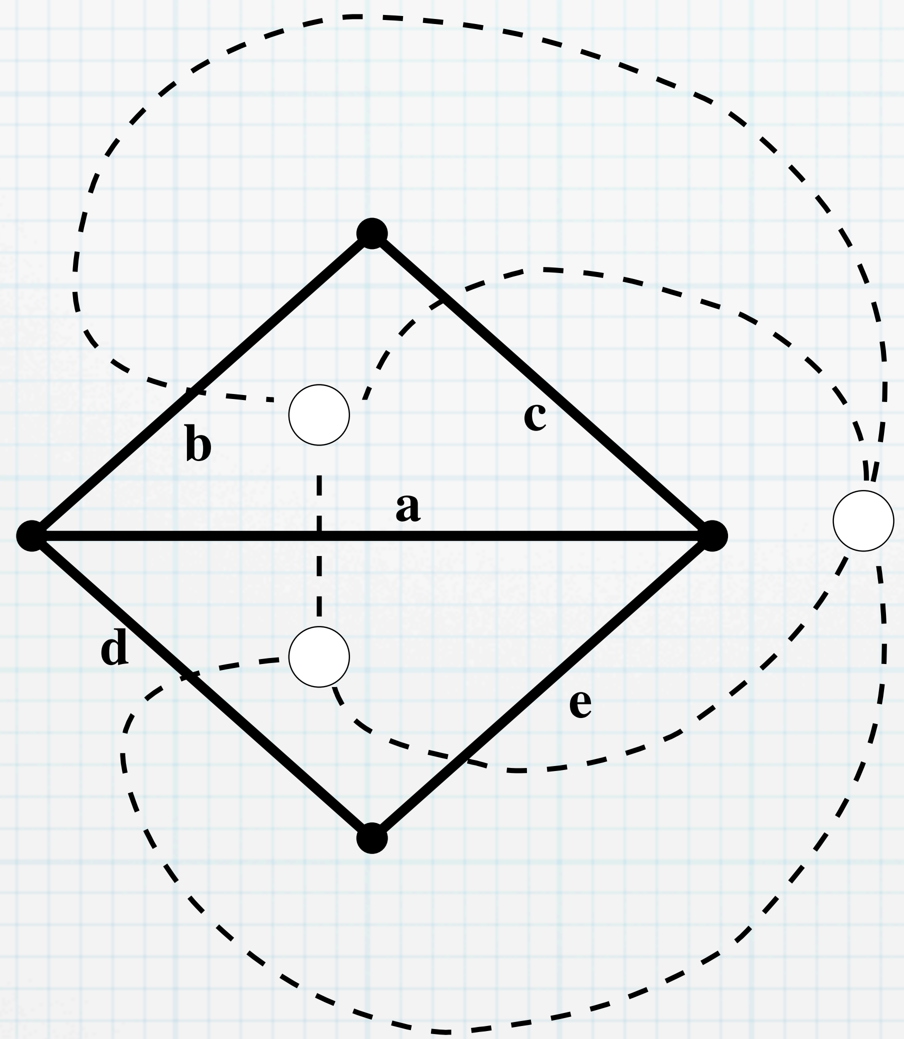
Use of new framework for approximation schemes for planar graphs

Problem	Time	efficient?
• Traveling salesman [Klein, 2005]	$O(n)$	✓
• Traveling salesman on subset of vertices [Klein, 2006]	$O(n \log n)$	✓
• 2-edge-connected spanning subgraph [Berger, Grigni, 2007]	unit-weights: $O(n)$	✓
	general weights: $O(n^{f(\varepsilon)})$	✗
• Steiner tree [Borradaile, Klein, Mathieu, 2008]	$O(n \log n)$	✓
• 2-edge-connected Steiner multisubgraph [Borradaile, Klein, 2008]	$O(n \log n)$	✓
• Steiner forest [Bateni, Hajiaghayi, Marx, 2010]	$O(n^{f(\varepsilon)})$	✗
speed-up [Eisenstat et al., new]	$O(n \log^{f(\varepsilon)} n)$	✓
• Prize-collecting Steiner tree, TSP, stroll [Bateni, Chekuri, Ene, Hajiaghayi, Korula, Marx, 2011]	$O(n^c)$	✓
• Multiterminal cut [Bateni, Hajiaghayi, K., Mathieu, unpublished]	$O(n^c)$	✓

Framework generalized to broader graph classes

- Steiner tree in bounded-genus graphs [Borradaile, Demaine, Tazari, 2009]
- TSP in excluded-minor graphs [Demaine, Hajiaghayi, and Kawarabayashi, 2011]

Planar duality

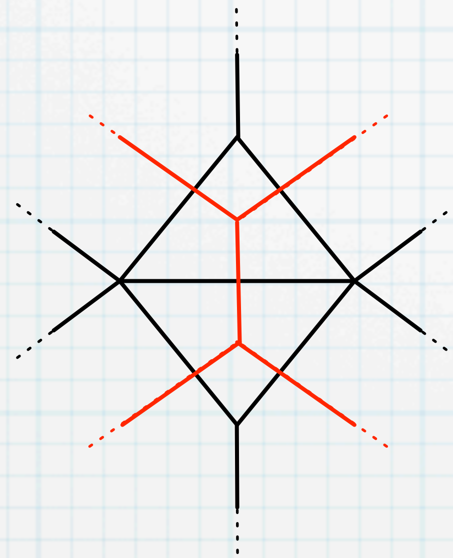


For each connected planar embedded graph, the *dual* is another connected planar embedded graph:

- Dual has a vertex for each face of the *primal* (the original graph)
- Dual has an edge for each edge of the primal.

One key idea for framework

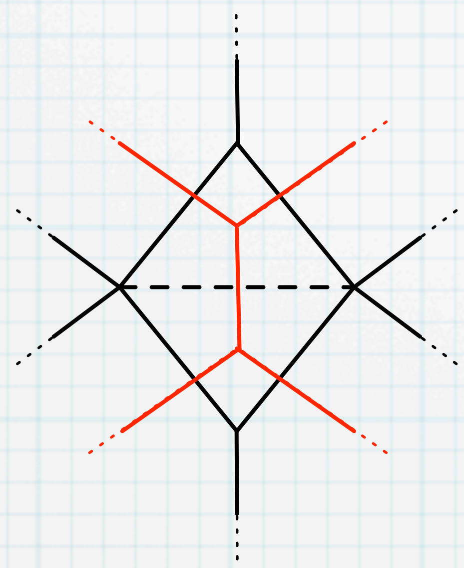
Deletion and contraction* are dual to each other



Deletion of a (non-self-loop) edge in the primal corresponds to contraction in the dual and vice versa

One key idea for framework

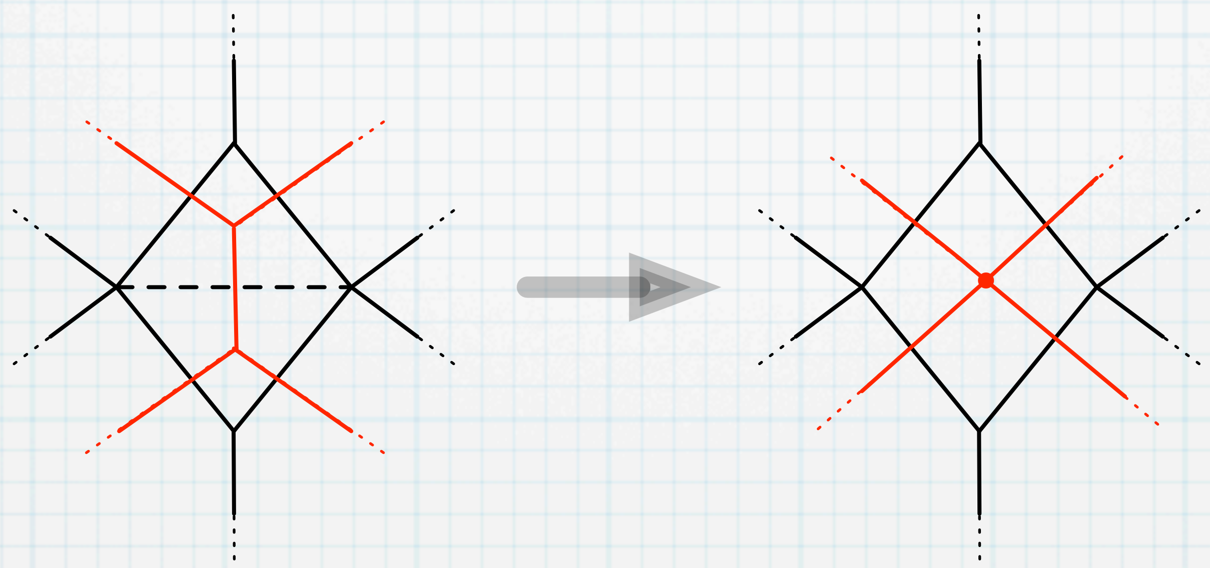
Deletion and contraction* are dual to each other



Deletion of a (non-self-loop) edge in the primal corresponds to contraction in the dual and vice versa

One key idea for framework

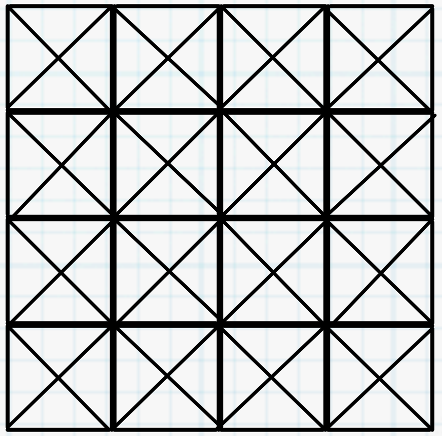
Deletion and contraction* are dual to each other



Deletion of a (non-self-loop) edge in the primal
corresponds to contraction in the dual
and vice versa

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor



Ensure total cost of resulting graph is *O(OPT)*

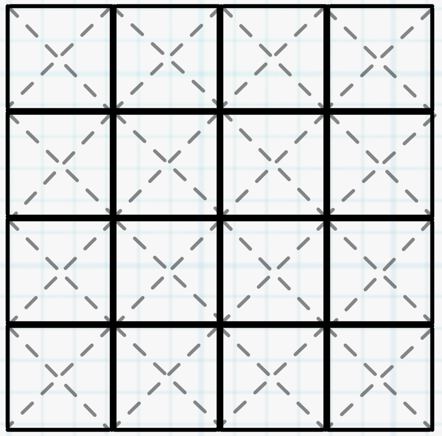
2. *Contract* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by $1/p \times O(OPT)$

Framework for approximation schemes for planar graphs [Klein, 2005]



1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

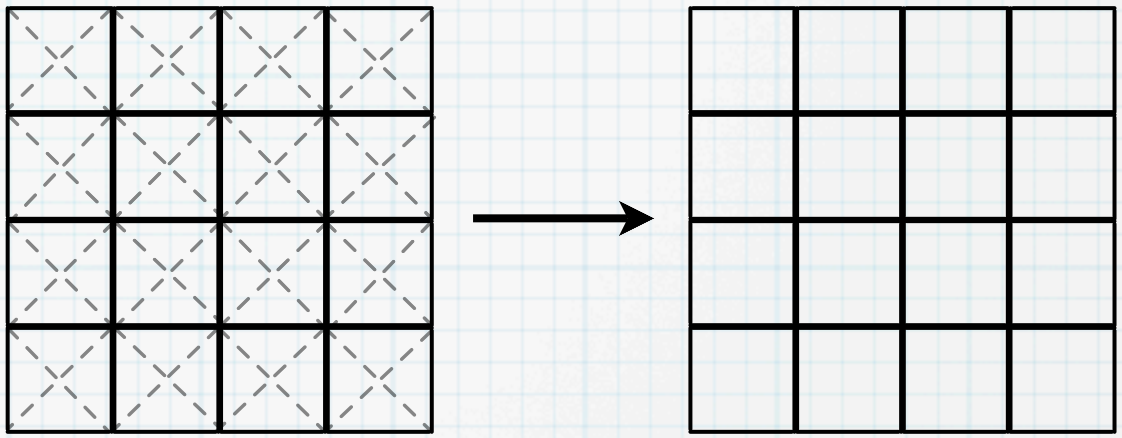
Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by 1/*p* × *O(OPT)*

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor



Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

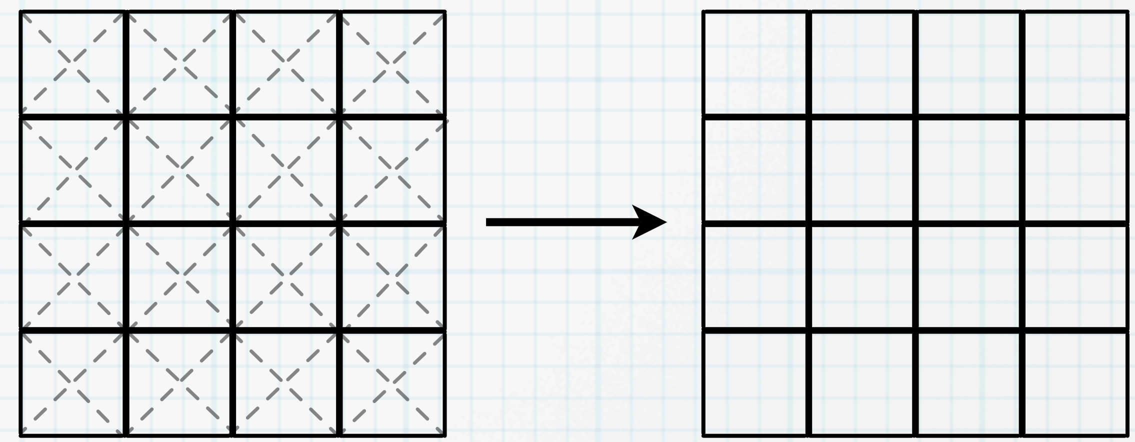
Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by 1/*p* × *O(OPT)*

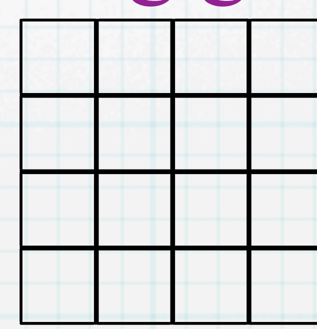
Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor



Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total



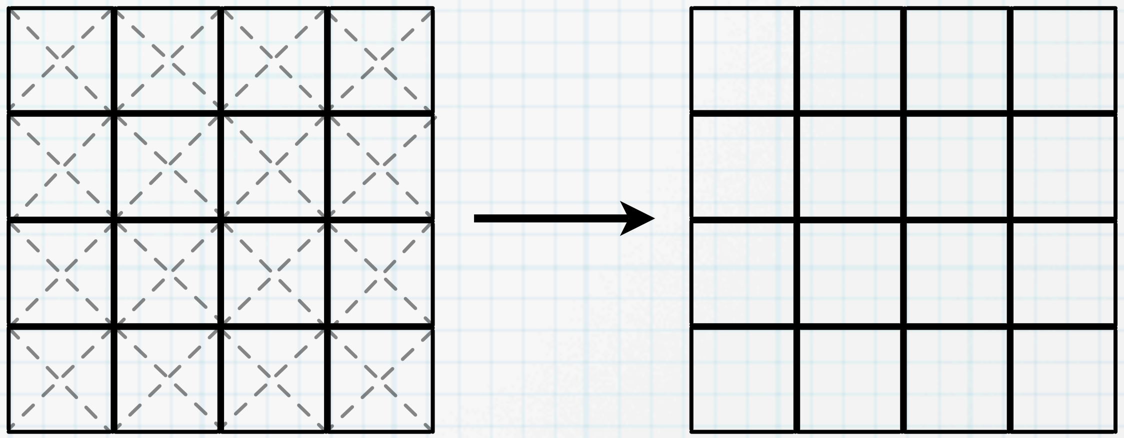
Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by 1/*p* × *O(OPT)*

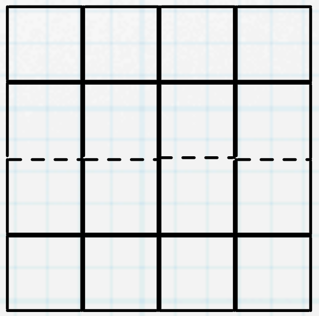
Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor



Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total



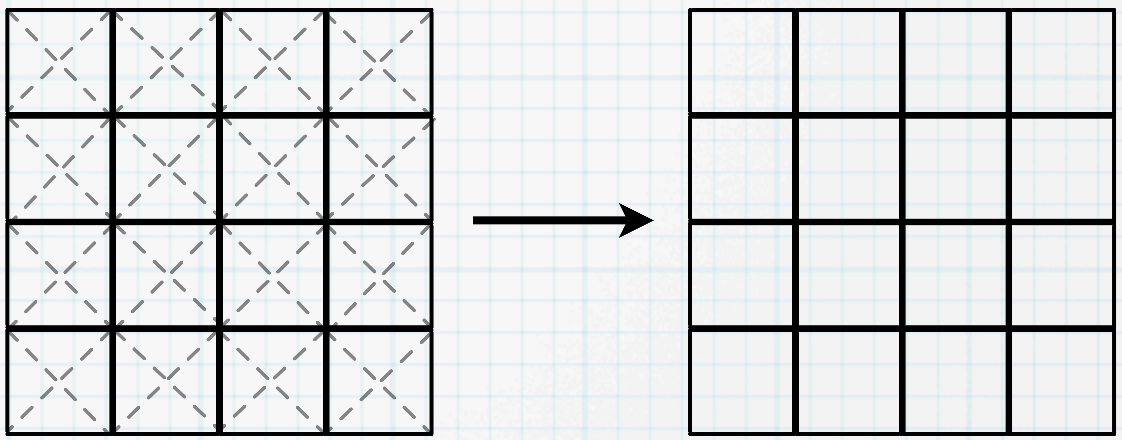
Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by 1/*p* × *O(OPT)*

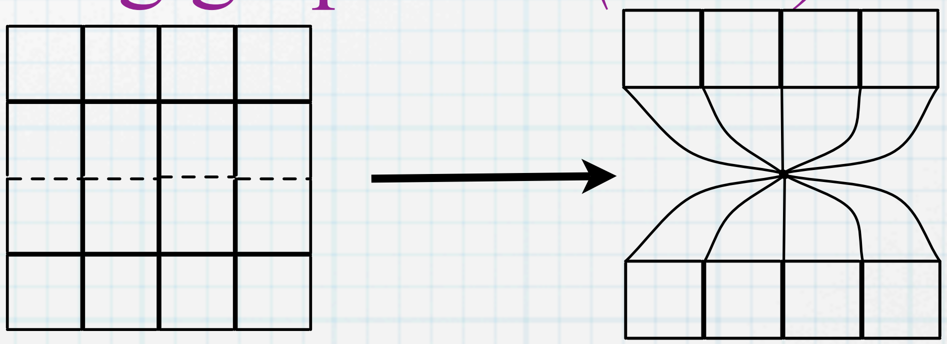
Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor



Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total



Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by $1/p \times O(OPT)$

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by $1/p \times O(OPT)$

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than $1+\varepsilon$ factor

1. *Contract* some edges while keeping OPT from increasing by more than $1+\varepsilon$ factor

Ensure total cost of resulting graph is $O(OPT)$

2. *Contract* edges of total cost at most $1/p$ times total

2. *Delete* edges of total cost at most $1/p$ times total

Ensure resulting graph has branchwidth $O(p)$

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by $1/p \times O(OPT)$

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor	1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total	2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

4. Lift solution to original graph, increasing cost by $1/p \times O(OPT)$

Choose p big enough so increase is $\leq \varepsilon$ *OPT*

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

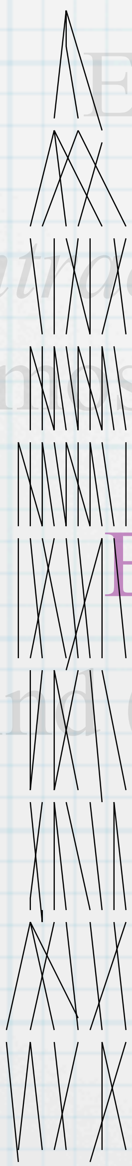
2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

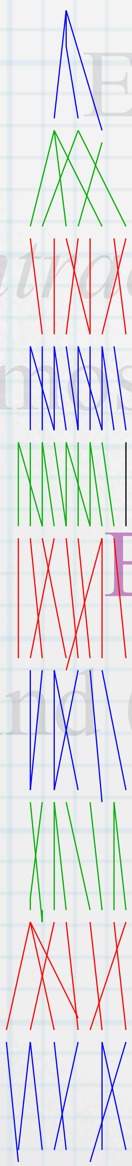


Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*



2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

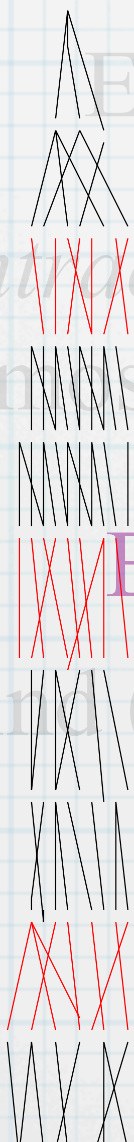
Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph



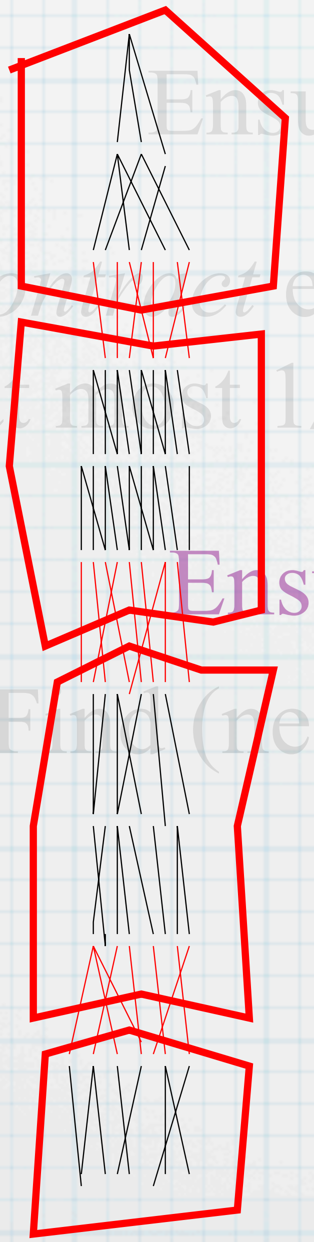
This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*



2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Framework for approximation schemes for planar graphs [Klein, 2005]

1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

2. *Contract* edges of total cost at most 1/*p* times total

2. *Delete* edges of total cost at most 1/*p* times total

Ensure resulting graph has branchwidth *O(p)*

3. Find (near-)optimal solution in low-branchwidth graph

This is just deleting in the planar dual. (In next talk, same idea in larger graph classes.)

This was known before, implicit in Baker's work. For planar graphs, do breadth-first search, p-color the levels, and delete the cheapest level.

Key step for most problems: "spanner" construction

1. *Delete* some edges while keeping OPT from increasing by more than $1+\varepsilon$ factor

1. *Contract* some edges while keeping OPT from increasing by more than $1+\varepsilon$ factor

Ensure total cost of resulting graph is *O(OPT)*

Key step for most problems: "spanner" construction

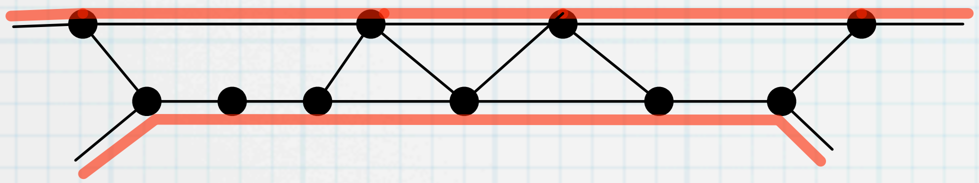
1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

Traveling salesman problem:

How to ensure that the resulting graph approximately preserves OPT?



Key step for most problems: "spanner" construction

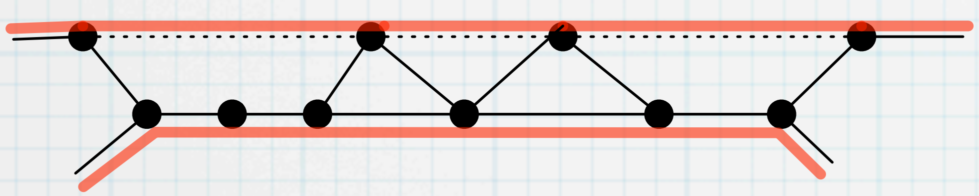
1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

Traveling salesman problem:

How to ensure that the resulting graph approximately preserves OPT?



Key step for most problems: “spanner” construction

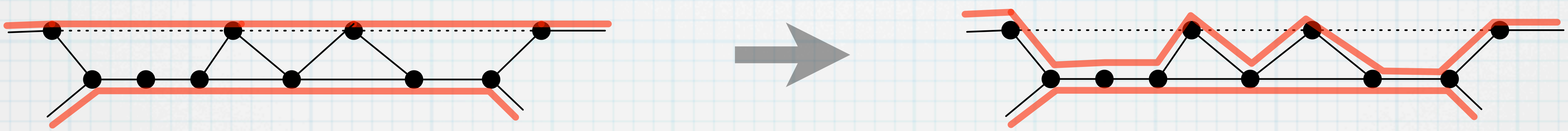
1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

Traveling salesman problem:

How to ensure that the resulting graph approximately preserves OPT?



Key step for most problems: “spanner” construction

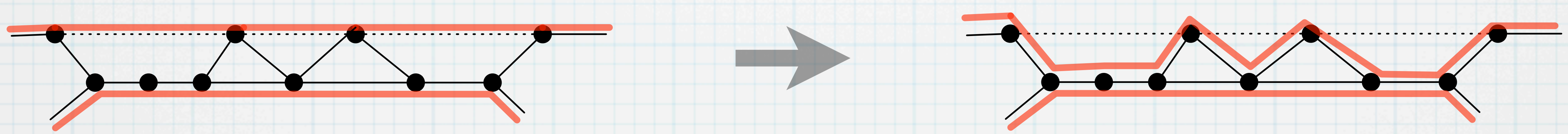
1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

Traveling salesman problem:

How to ensure that the resulting graph approximately preserves OPT?



Consider optimal tour. Replace each edge by a 1+ε-approximate shortest path. Resulting tour is 1+ε-approximate.

Key step for most problems: "spanner" construction

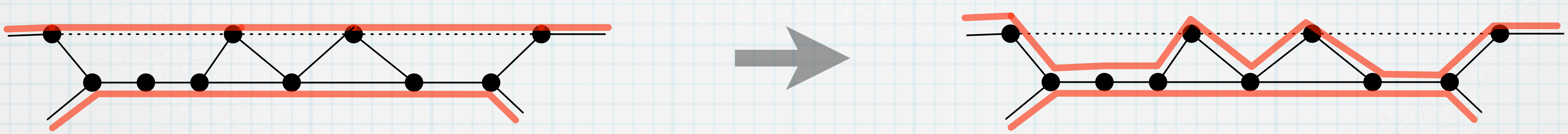
1. *Delete* some edges while keeping OPT from increasing by more than 1+ε factor

1. *Contract* some edges while keeping OPT from increasing by more than 1+ε factor

Ensure total cost of resulting graph is *O(OPT)*

Traveling salesman problem:

How to ensure that the resulting graph approximately preserves OPT?



Consider optimal tour. Replace each edge by a 1+ε-approximate shortest path. Resulting tour is 1+ε-approximate.

Therefore: it suffices to select a subset of edges that approximately preserves vertex-to-vertex distances.

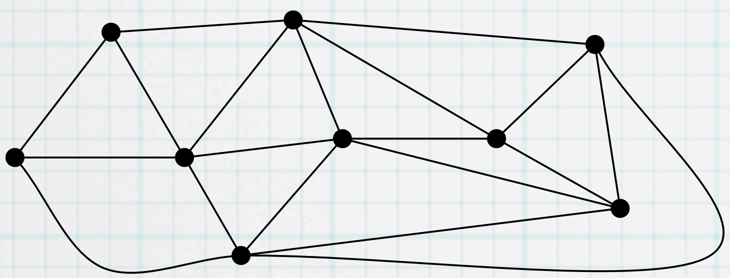
Selecting a low-weight subset of edges that approximately preserves vertex-to-vertex distances

O(n^2) time [Althoffer, Das, Dobkin, Joseph, Soares, 1993], linear time [Klein, 2005]

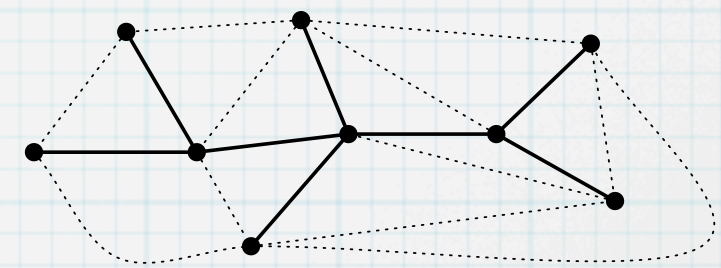
Just achieving finite distances requires a spanning tree.

To keep weight low, start with *minimum-weight* spanning tree (MST).

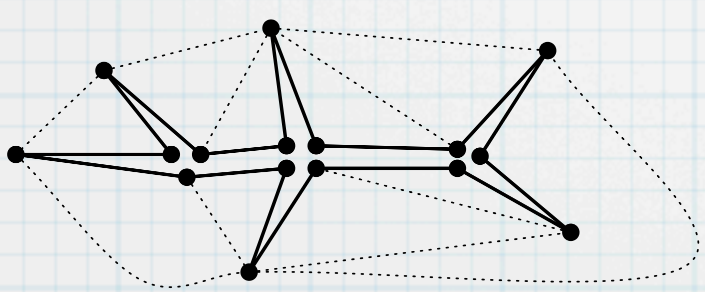
Will choose additional edges of total weight ≤ (2/ε) *weight(MST)*.

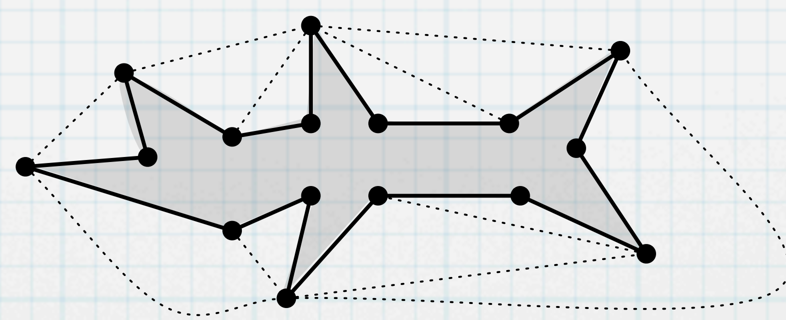


Step 1: Let *T* be the minimum-weight spanning tree. Include it in the spanner.

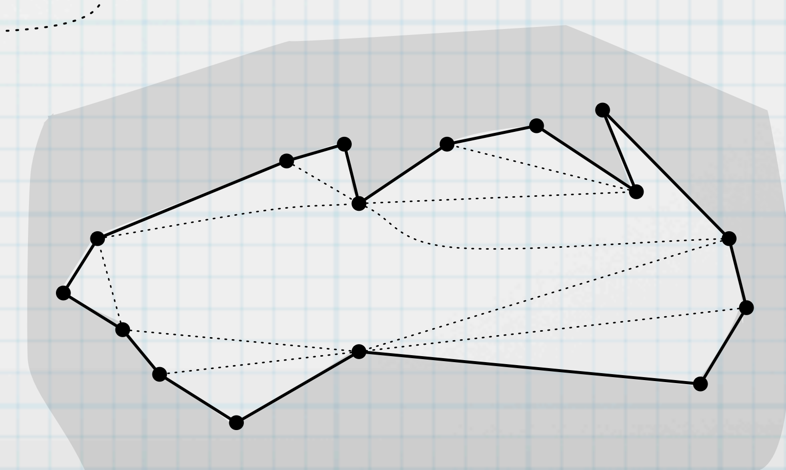


Step 2: Cut along T, duplicating edges and vertices.





Step 3: Consider resulting face as infinite face.



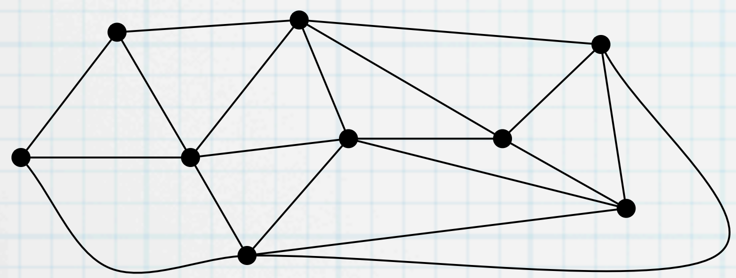
Selecting a low-weight subset of edges that approximately preserves vertex-to-vertex distances

$O(n^2)$ time [Althoffer, Das, Dobkin, Joseph, Soares, 1993], linear time [Klein, 2005]

Just achieving finite distances requires a spanning tree.

To keep weight low, start with *minimum-weight* spanning tree (MST).

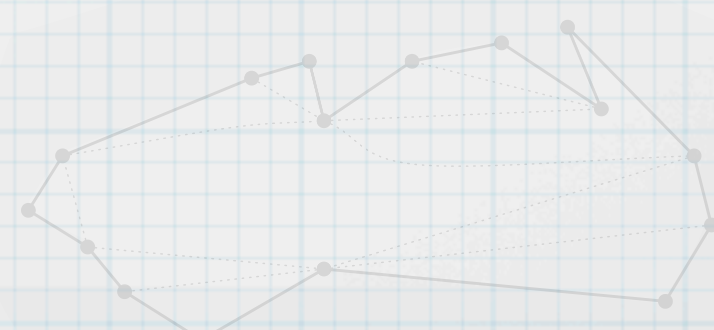
Will choose additional edges of total weight $\leq$ $(2/\varepsilon)$ *weight(MST)*.



Step 1: Let T be the minimum-weight spanning tree. Include it in the spanner.

Step 2: Cut along T, duplicating edges and vertices.

Step 3: Consider resulting face as infinite face.



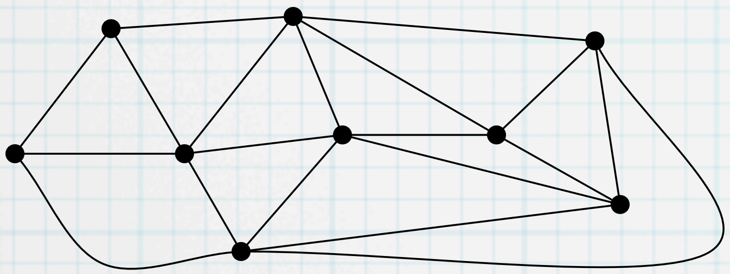
Selecting a low-weight subset of edges that approximately preserves vertex-to-vertex distances

$O(n^2)$ time [Althoffer, Das, Dobkin, Joseph, Soares, 1993], linear time [Klein, 2005]

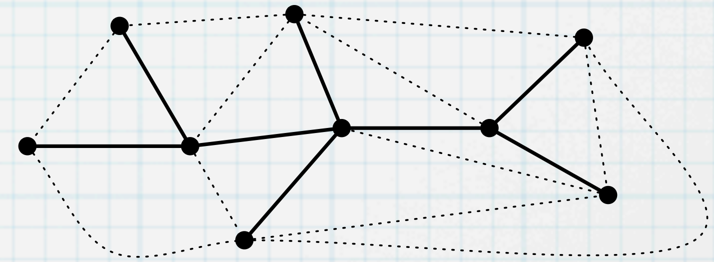
Just achieving finite distances requires a spanning tree.

To keep weight low, start with *minimum-weight* spanning tree (MST).

Will choose additional edges of total weight $\leq$ $(2/\varepsilon)$ *weight(MST)*.

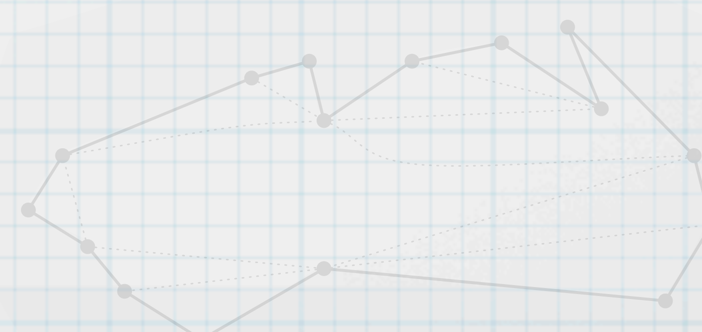


Step 1: Let *T* be the minimum-weight spanning tree. Include it in the spanner.



Step 2: Cut along T, duplicating edges and vertices.

Step 3: Consider resulting face as infinite face.



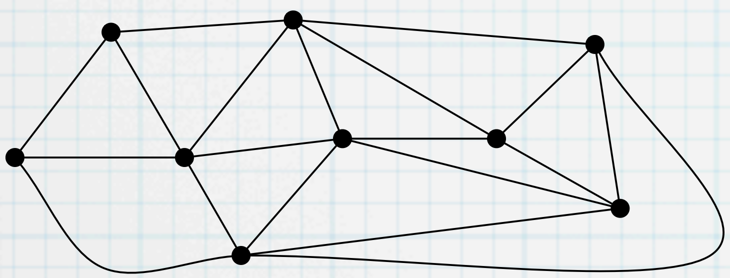
Selecting a low-weight subset of edges that approximately preserves vertex-to-vertex distances

O(n^2) time [Althoffer, Das, Dobkin, Joseph, Soares, 1993], linear time [Klein, 2005]

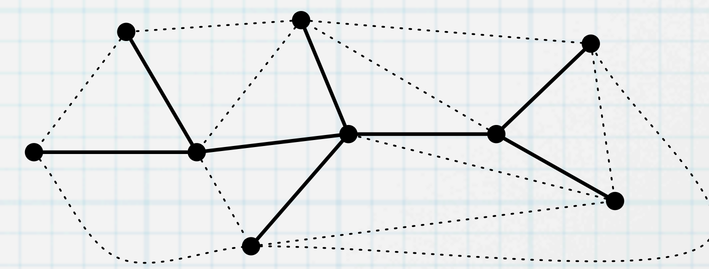
Just achieving finite distances requires a spanning tree.

To keep weight low, start with *minimum-weight* spanning tree (MST).

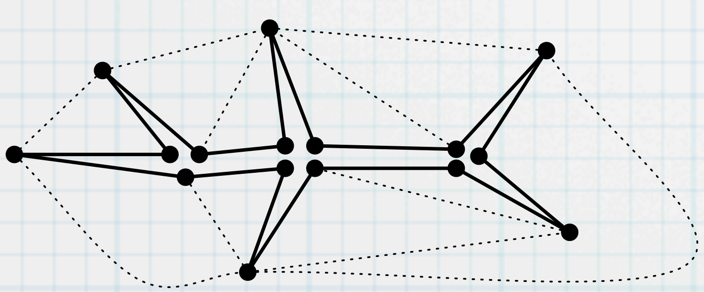
Will choose additional edges of total weight ≤ $(2/\varepsilon)$ *weight(MST)*.

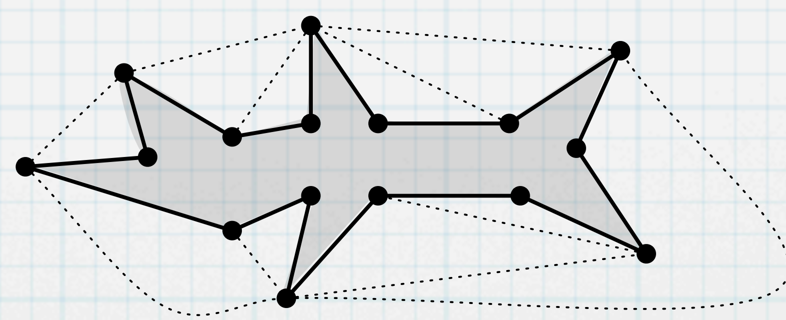


Step 1: Let *T* be the minimum-weight spanning tree. Include it in the spanner.

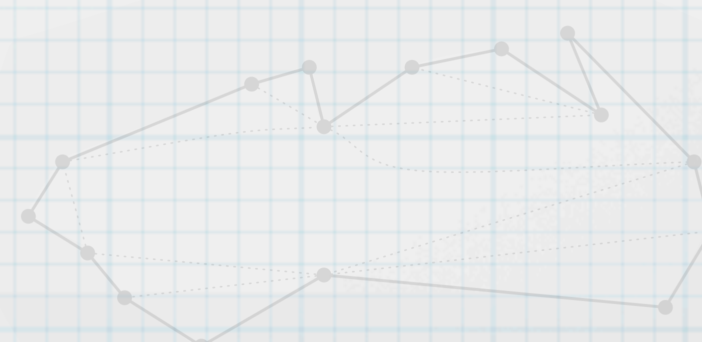


Step 2: Cut along T, duplicating edges and vertices.





Step 3: Consider resulting face as infinite face.



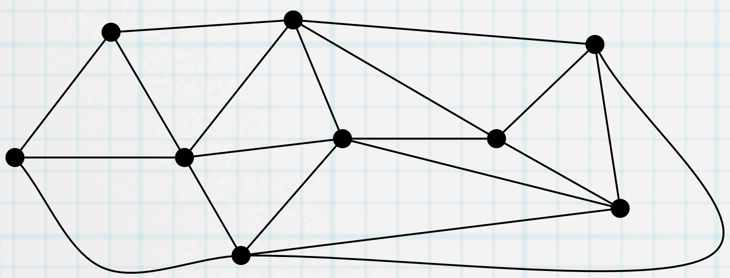
Selecting a low-weight subset of edges that approximately preserves vertex-to-vertex distances

$O(n^2)$ time [Althoffer, Das, Dobkin, Joseph, Soares, 1993], linear time [Klein, 2005]

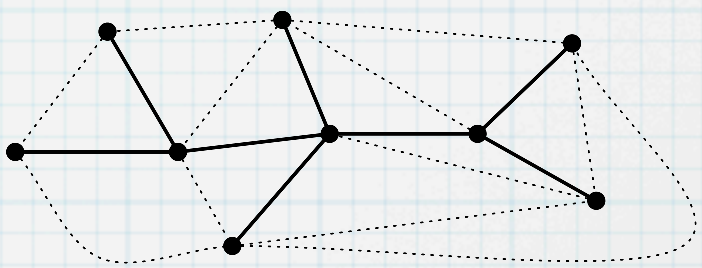
Just achieving finite distances requires a spanning tree.

To keep weight low, start with *minimum-weight* spanning tree (MST).

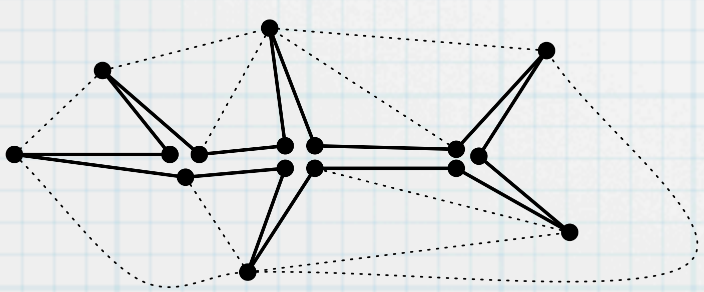
Will choose additional edges of total weight $\leq$ $(2/\varepsilon)$ *weight(MST)*.

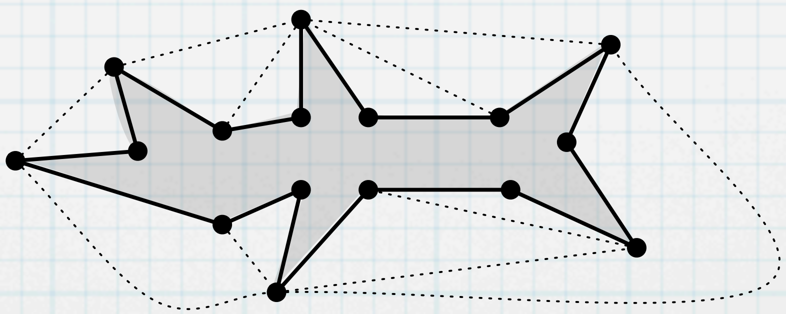


Step 1: Let T be the minimum-weight spanning tree. Include it in the spanner.

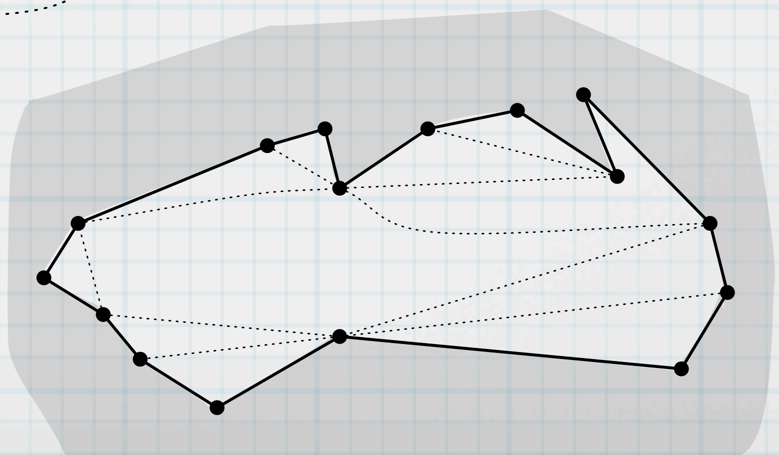


Step 2: Cut along T, duplicating edges and vertices.

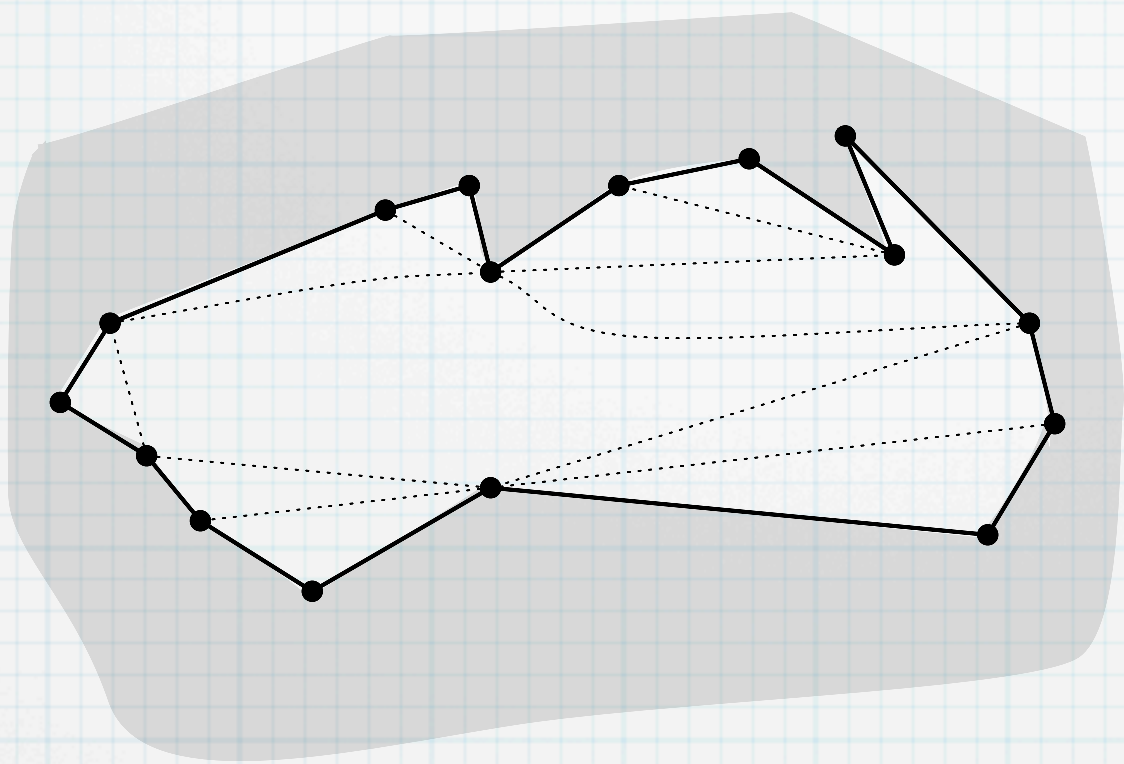




Step 3: Consider resulting face as infinite face.



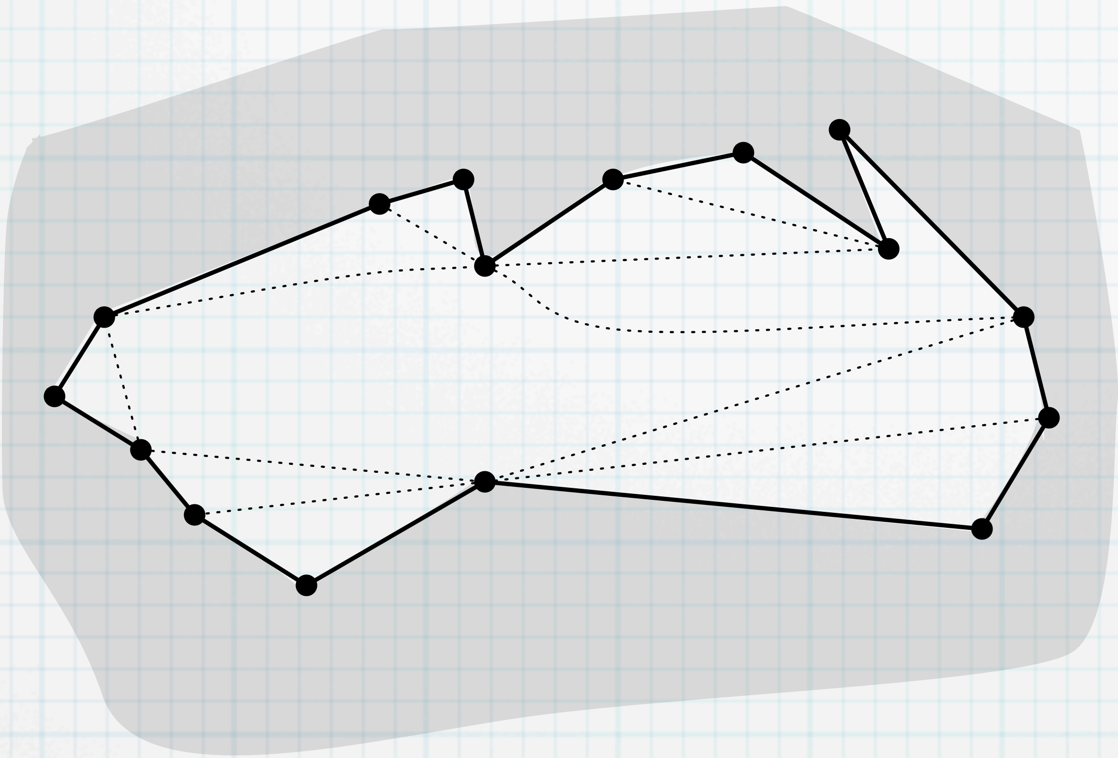
Step 4: Consider nontree edges in order.
For each such edge uv, if
$$(1+\varepsilon)\ \mathrm{weight}(uv) \le \text{weight of corresponding boundary subpath}$$
then add uv to spanner and chop along uv



For each edge uv added to spanner, boundary weight goes down by at least
$$\varepsilon\ \mathit{weight}(uv)$$
Therefore, total weight added to spanner is at most
$$\varepsilon^{-1} \cdot \textit{decrease in boundary weight}$$

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
$$2\ \varepsilon^{-1}\ \mathit{weight}(MST)$$

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

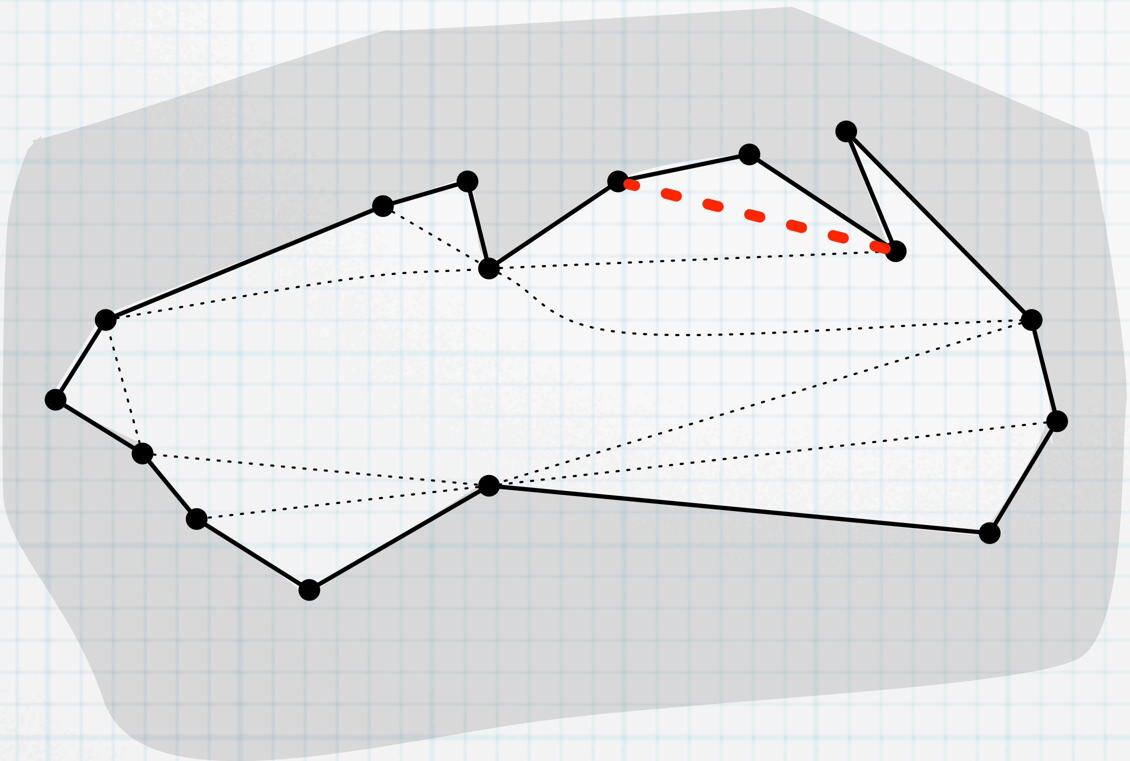
Therefore, total weight added to spanner is at most
$\varepsilon^{-1} \cdot$ *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
$2\,\varepsilon^{-1}$ *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if

$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath

then add *uv* to spanner and chop along *uv*

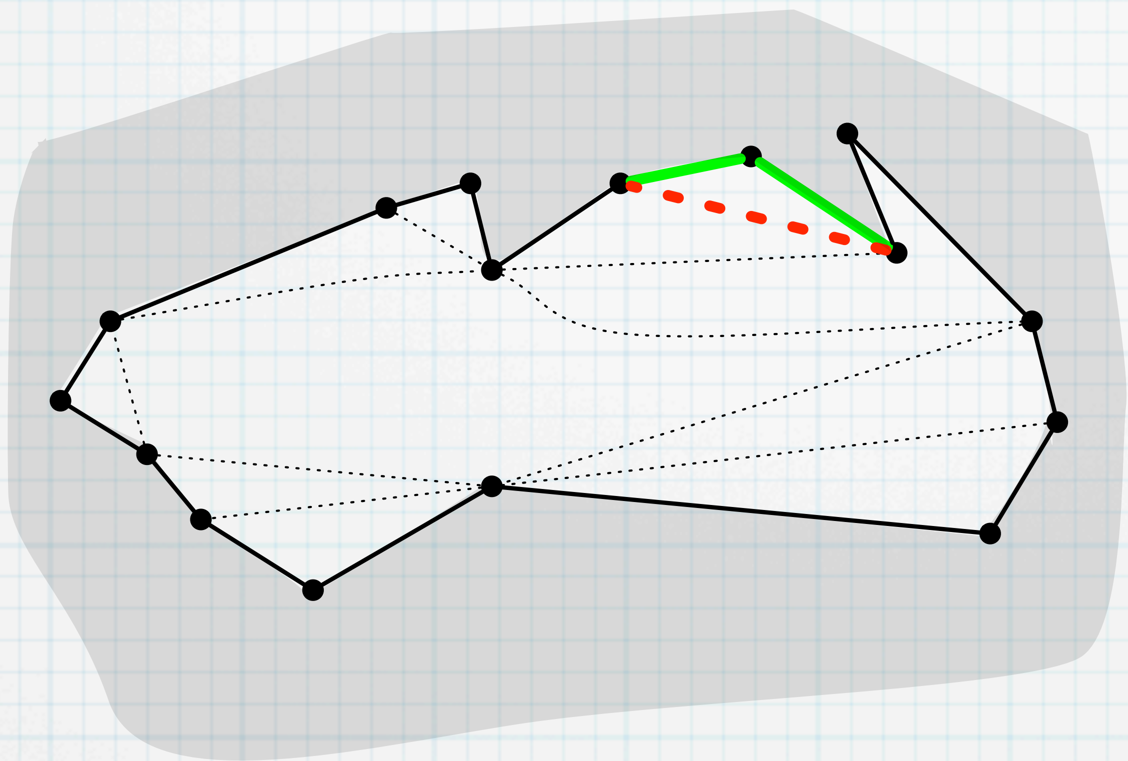


For each edge *uv* added to spanner, boundary weight goes down by at least *ε weight(uv)*

Therefore, total weight added to spanner is at most *ε^{-1} · decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is 2 *ε^{-1} weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

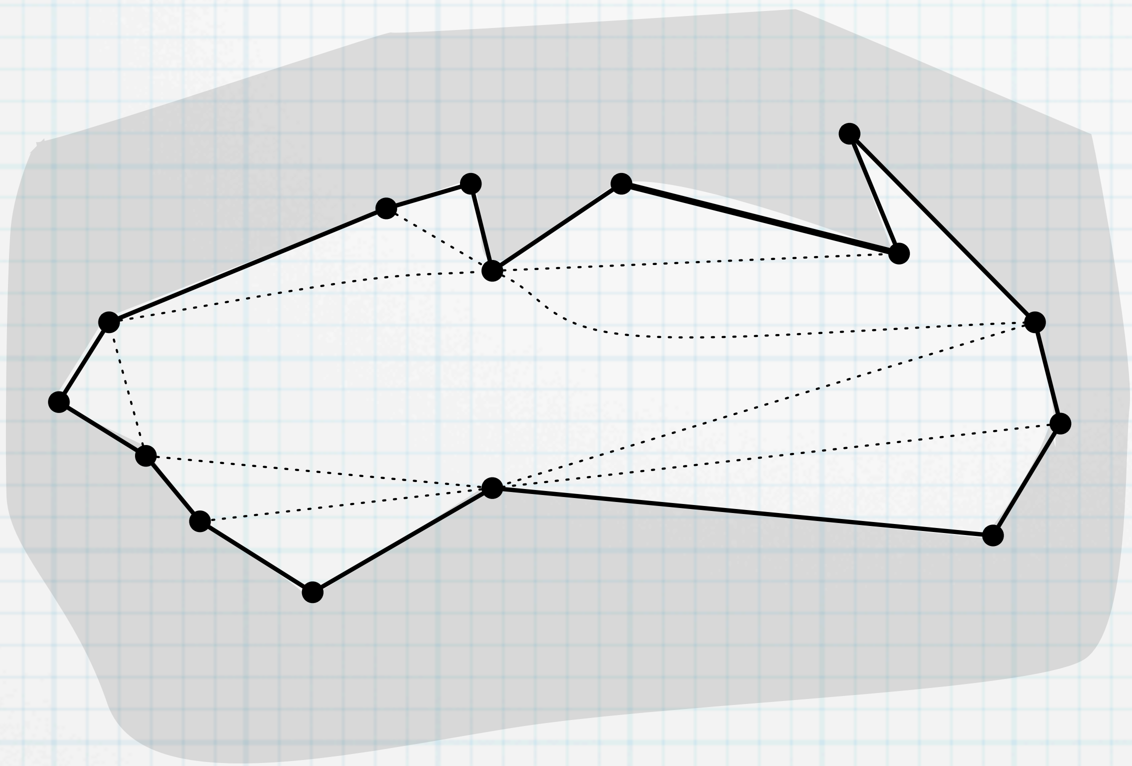
Therefore, total weight added to spanner is at most
$\varepsilon^{-1} \cdot$ *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
$2\,\varepsilon^{-1}$ *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if

$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath

then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least

ε weight(uv)

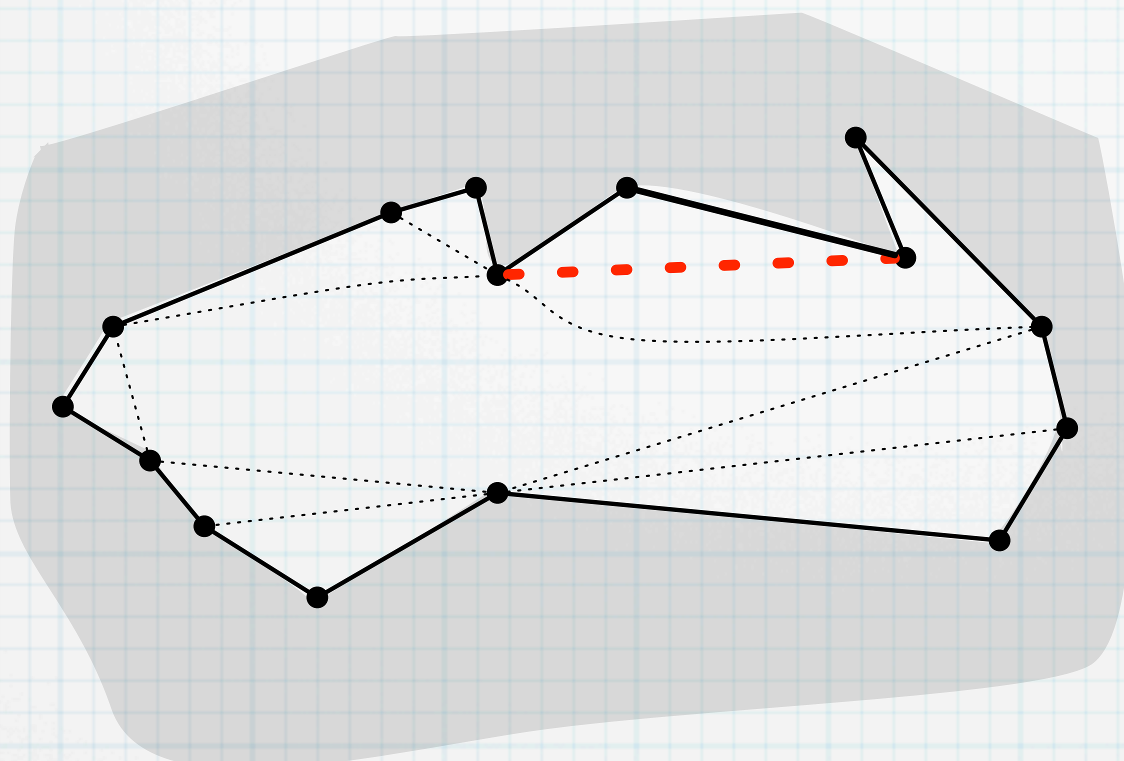
Therefore, total weight added to spanner is at most

$\varepsilon^{-1} \cdot$ *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is

2 ε^{-1} *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*

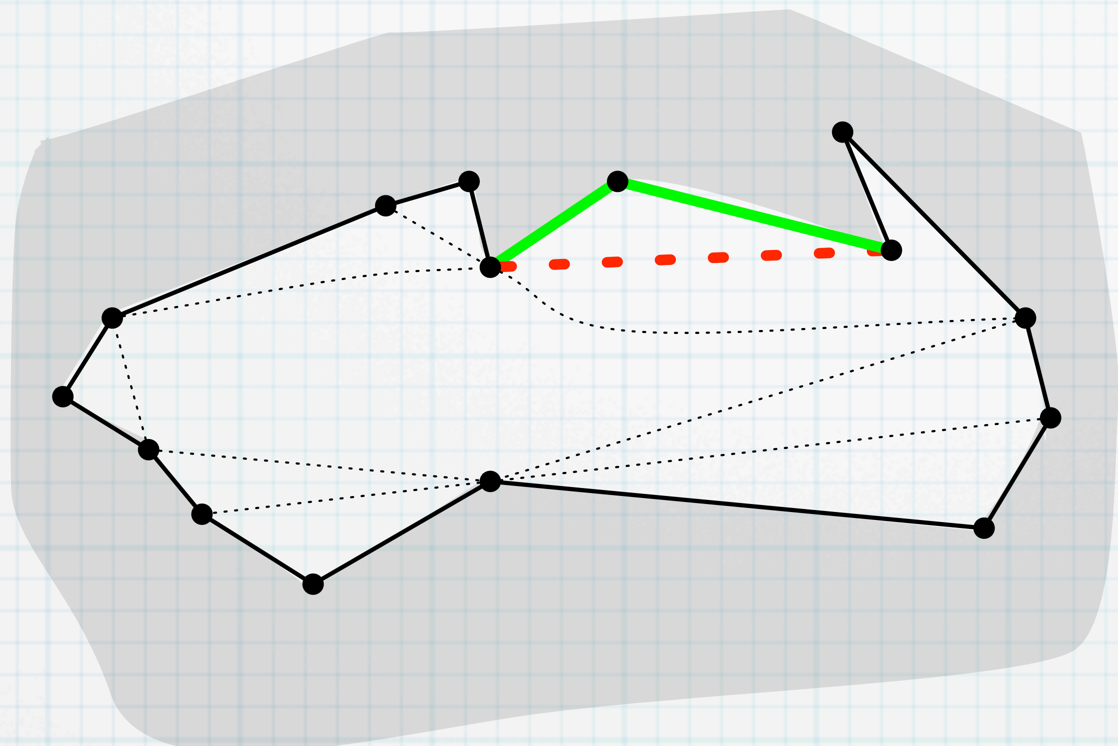


For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

Therefore, total weight added to spanner is at most
ε^{-1} · decrease in boundary weight

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
2 *ε^{-1} weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

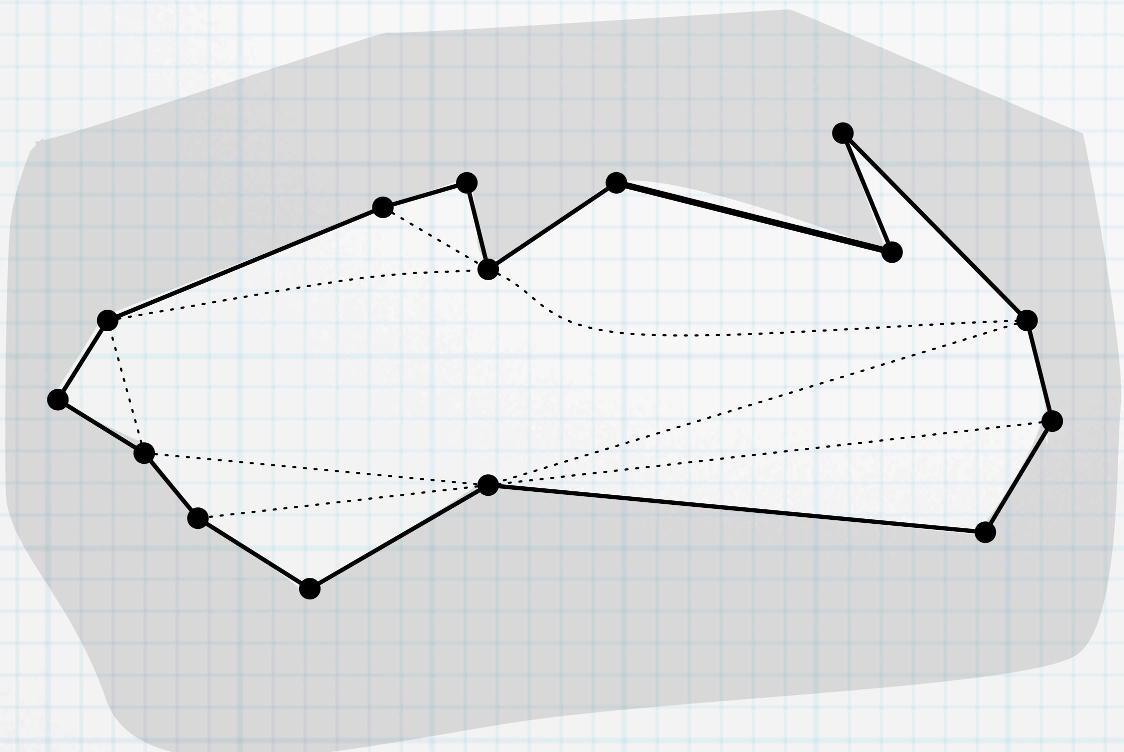
Therefore, total weight added to spanner is at most
ε^{-1} · *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
2 ε^{-1} *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if

$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath

then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least

ε weight(uv)

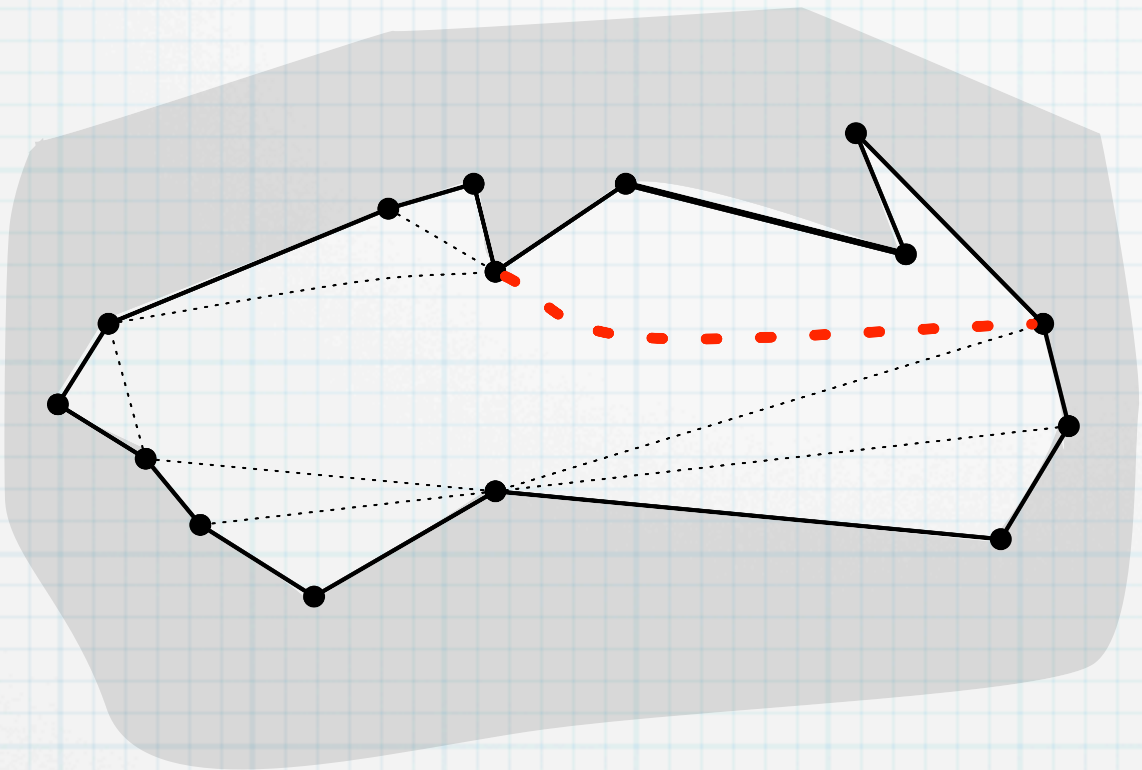
Therefore, total weight added to spanner is at most

$\varepsilon^{-1} \cdot$ *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is

$2\,\varepsilon^{-1}$ *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
(1+ε) weight(*uv*) ≤ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*

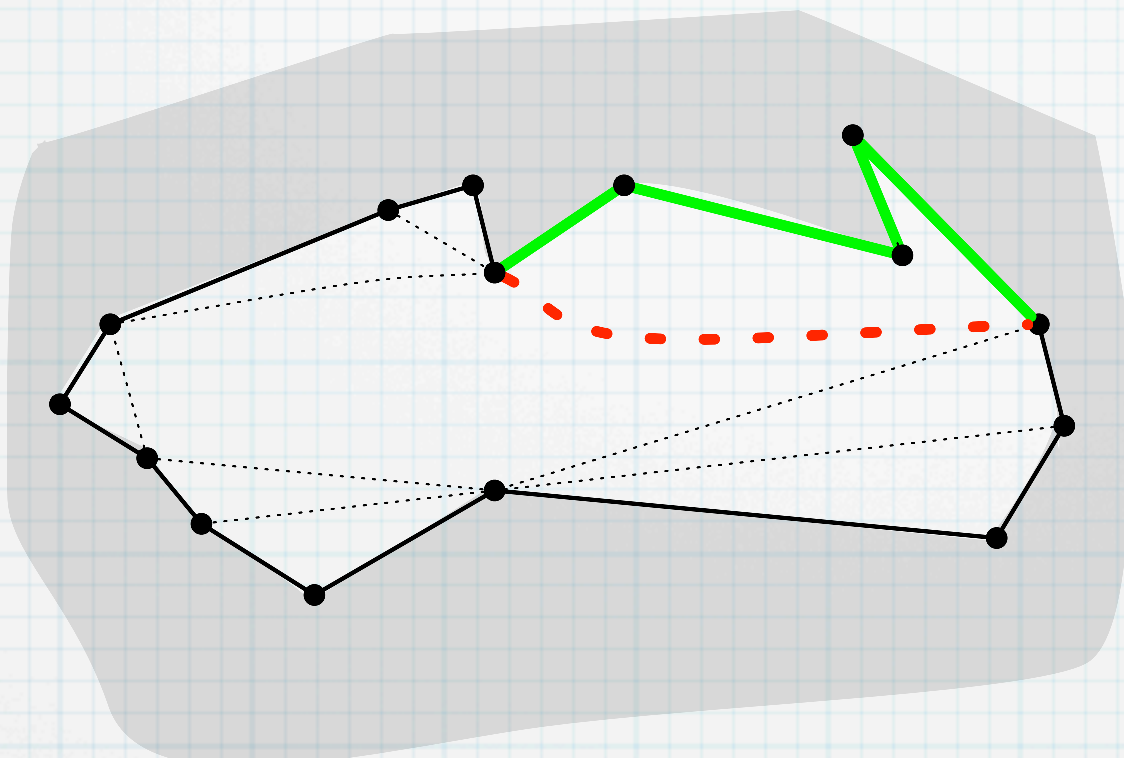


For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

Therefore, total weight added to spanner is at most
ε⁻¹ · decrease in boundary weight

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
2 ε⁻¹ weight(MST)

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
(1+ε) weight(*uv*) ≤ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*

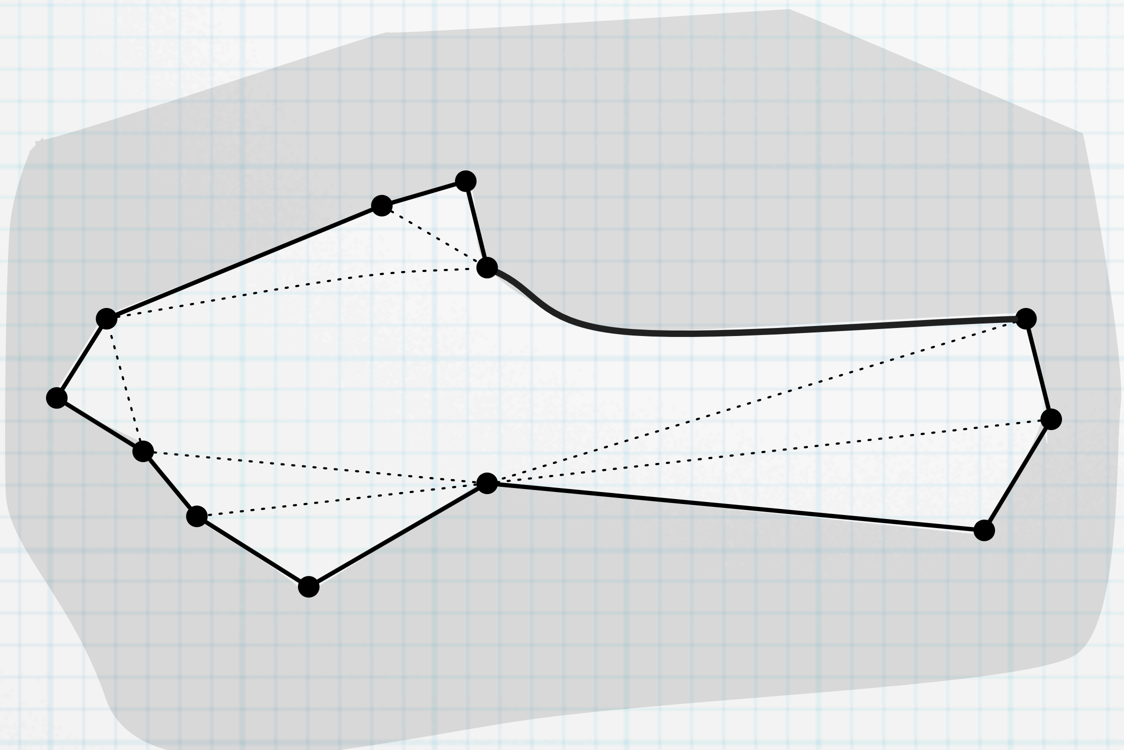


For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

Therefore, total weight added to spanner is at most
ε^{-1} · decrease in boundary weight

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
2 ε^{-1} *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*

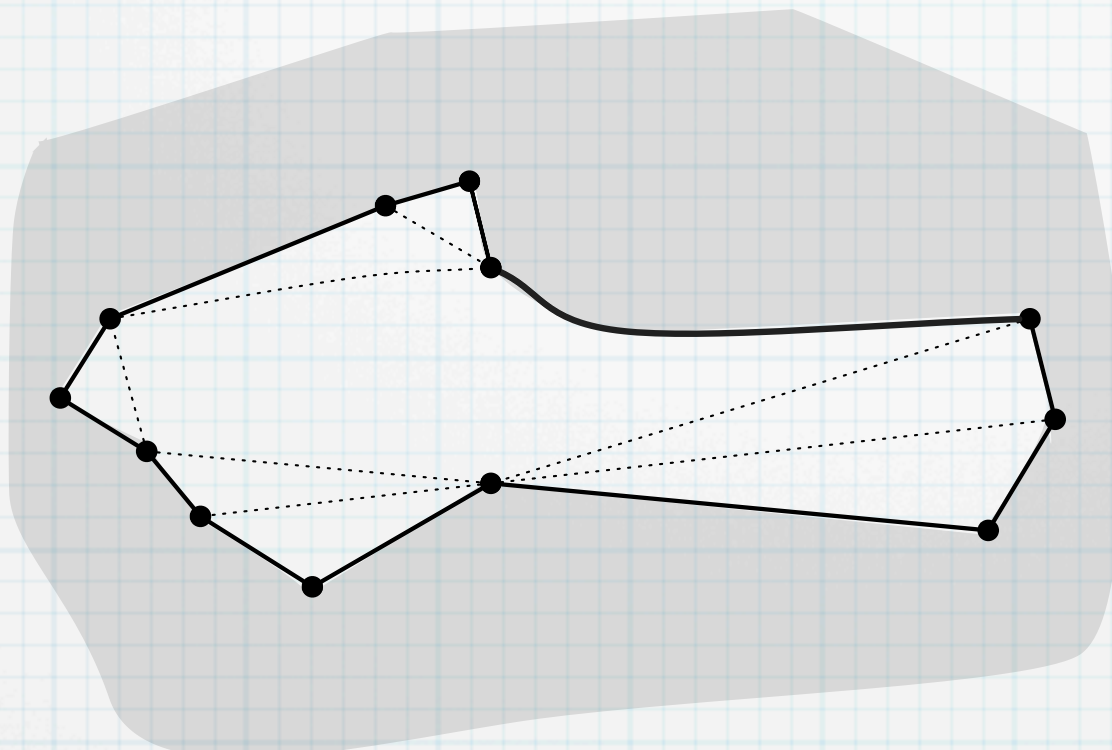


For each edge *uv* added to spanner, boundary weight goes down by at least
ε *weight(uv)*

Therefore, total weight added to spanner is at most
$\varepsilon^{-1} \cdot$ *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
$2\,\varepsilon^{-1}$ *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
(1+ε) weight(*uv*) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*

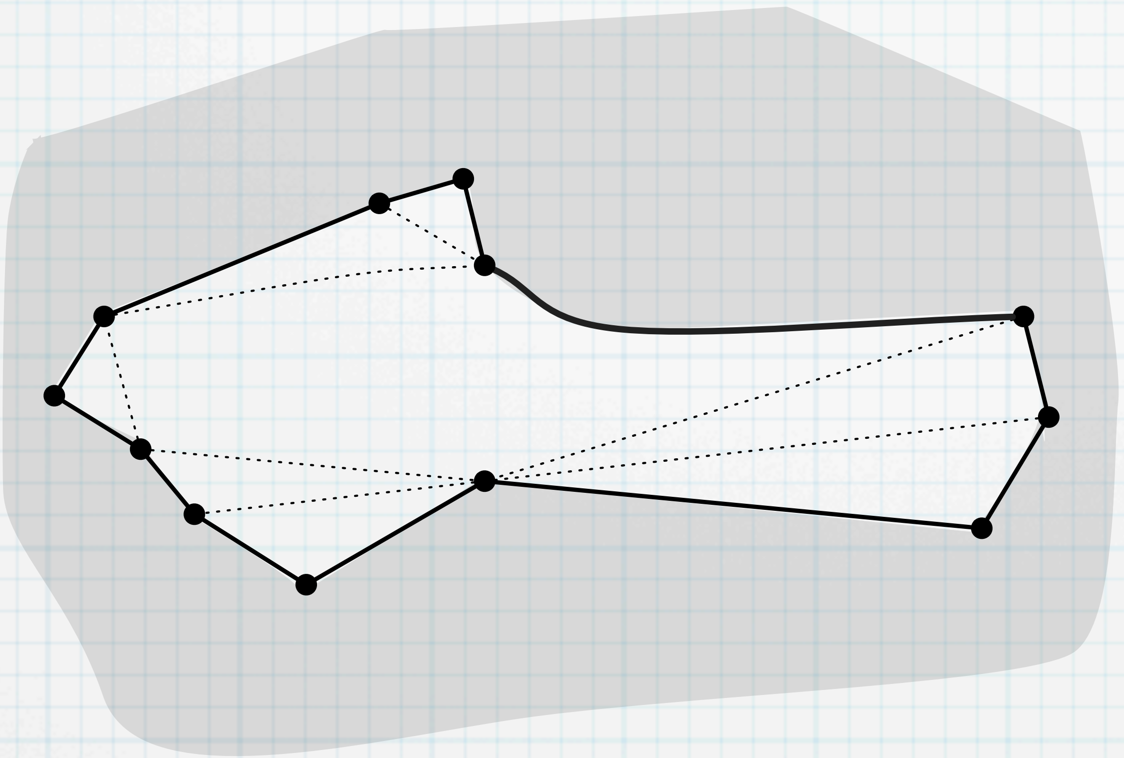


For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

Therefore, total weight added to spanner is at most
ε^{-1} · decrease in boundary weight

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
2 *ε^{-1} weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if
$(1+\varepsilon)$ weight(uv) $\leq$ weight of corresponding boundary subpath
then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least
ε weight(uv)

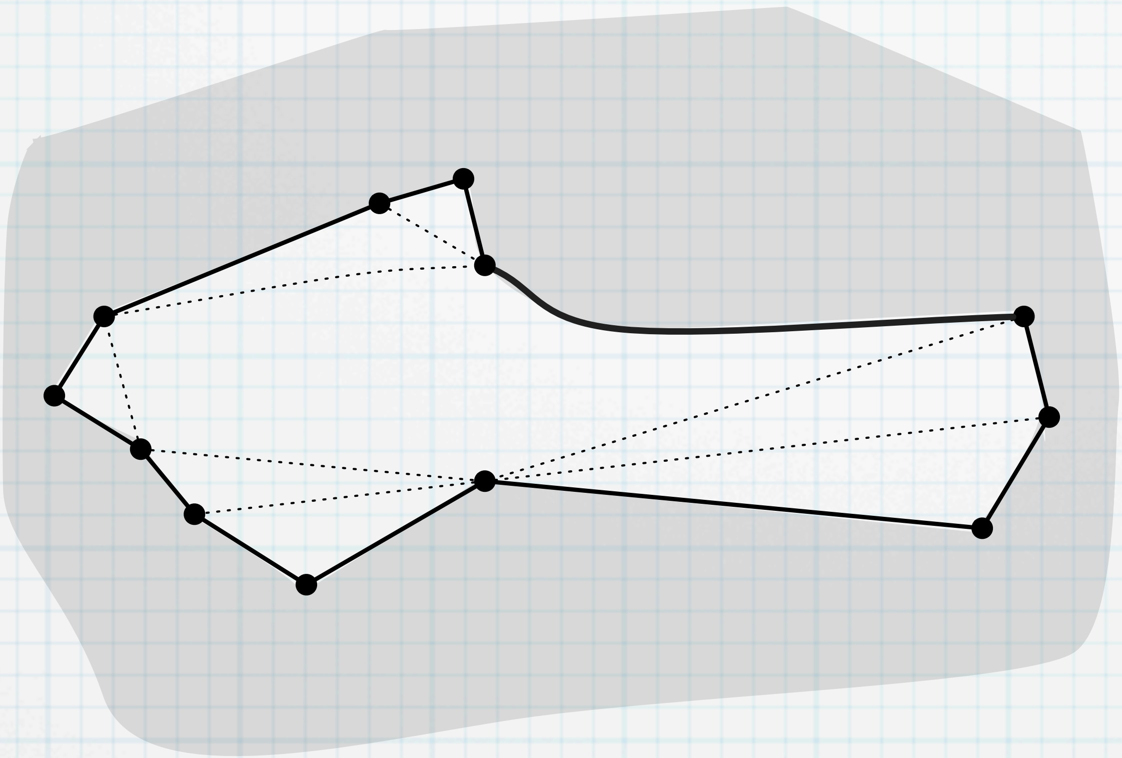
Therefore, total weight added to spanner is at most
ε^{-1} · *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is
2 ε^{-1} *weight(MST)*

Step 4: Consider nontree edges in order.
For each such edge *uv*, if

$(1+\varepsilon)$ weight(*uv*) $\leq$ weight of corresponding boundary subpath

then add *uv* to spanner and chop along *uv*



For each edge *uv* added to spanner, boundary weight goes down by at least

ε weight(uv)

Therefore, total weight added to spanner is at most

$\varepsilon^{-1} \cdot$ *decrease in boundary weight*

Initial boundary weight is 2 *weight(MST)*, so total weight added to spanner is

$2\ \varepsilon^{-1}$ *weight(MST)*

Theorem: for any undirected planar graph G with edge-weights, $\exists$ subgraph of cost $\leq 2(\varepsilon^{-1}+1) \times$ min spanning tree cost such that, $\forall u, v \in V$,
u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

In framework for approximation scheme, choose $p = \varepsilon / 2(\varepsilon^{-1}+1)$ so increase in cost is at most $\varepsilon\, OPT$

Corollary: There is a linear-time approximation scheme for traveling salesman in planar graphs.

Theorem: for any undirected planar graph G with edge-weights, $\exists$ subgraph of cost $\leq 2(\varepsilon^{-1}+1) \times$ min spanning tree cost such that, $\forall u,v \in V$,
u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

In framework for approximation scheme, choose $p = \varepsilon / 2(\varepsilon^{-1}+1)$ so increase in cost is at most ε OPT

Corollary: There is a linear-time approximation scheme for traveling salesman in planar graphs.

Theorem: for any undirected planar graph G with edge-weights, $\exists$ subgraph of cost $\leq 2(\varepsilon^{-1}+1) \times$ min spanning tree cost such that, $\forall u, v \in V$,
u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

In framework for approximation scheme, choose $p = \varepsilon / 2(\varepsilon^{-1}+1)$ so increase in cost is at most $\varepsilon\ OPT$

Corollary: There is a linear-time approximation scheme for traveling salesman in planar graphs.

Theorem: for any undirected planar graph G with edge-weights, $\exists$ subgraph of cost $\leq 2(\varepsilon^{-1}+1) \times$ min spanning tree cost such that, $\forall u,v \in V$, u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

In framework for approximation scheme, choose $p = \varepsilon / 2(\varepsilon^{-1}+1)$ so increase in cost is at most ε OPT

Corollary: There is a linear-time approximation scheme for traveling salesman in planar graphs.

Theorem: for any undirected planar graph G with edge-weights, $\exists$ subgraph of cost $\leq 2(\varepsilon^{-1}+1) \times$ min spanning tree cost such that, $\forall u,v \in V$,

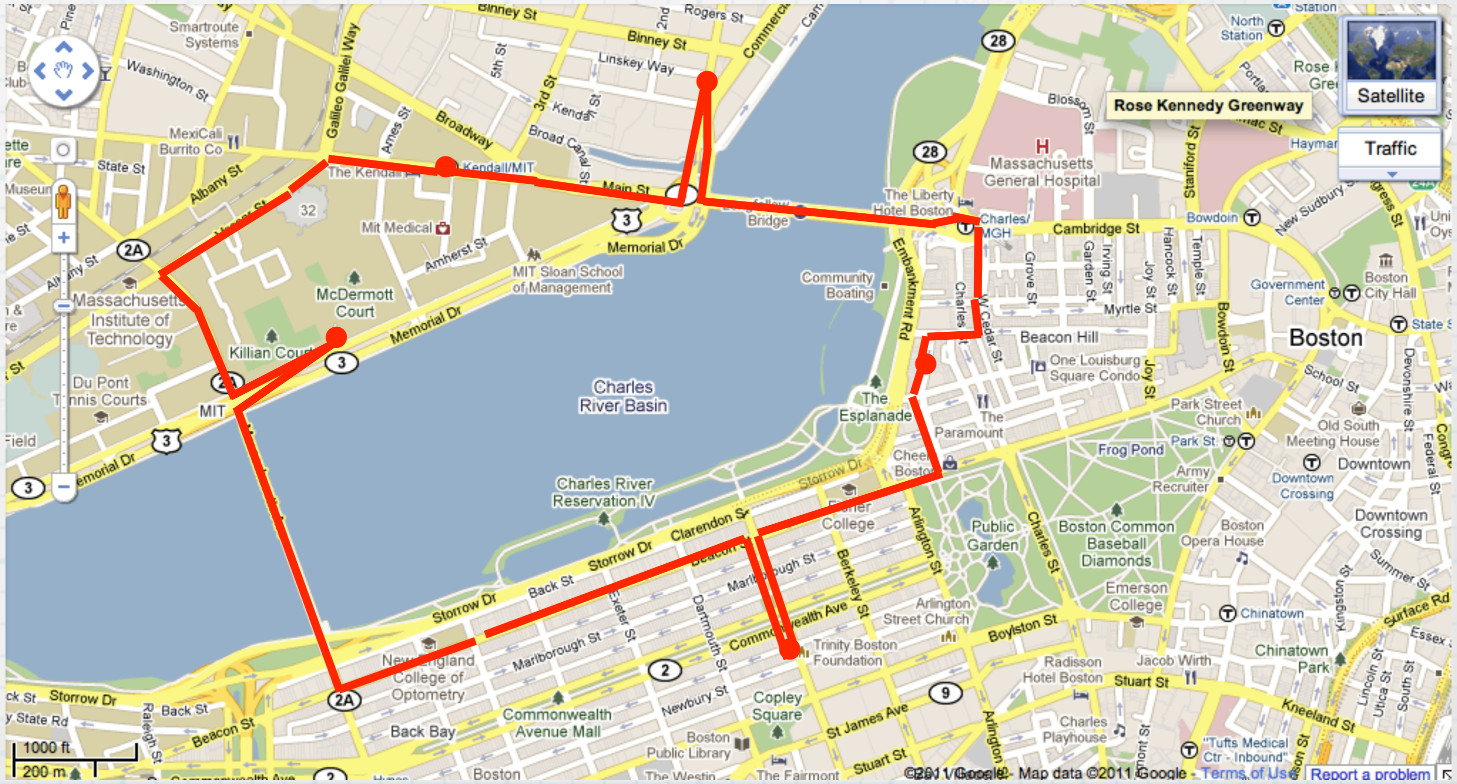
u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

In framework for approximation scheme, choose $p = \varepsilon / 2(\varepsilon^{-1}+1)$ so increase in cost is at most $\varepsilon\ OPT$

Corollary: There is a linear-time approximation scheme for traveling salesman in planar graphs.

But we want to address...

Traveling salesman on a *subset* of vertices



Theorem: for any undirected planar graph G with edge-weights, $\exists$ subgraph of cost $\leq 2(\varepsilon^{-1}+1) \times$ min spanning tree cost such that, $\forall u,v \in V$,

u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

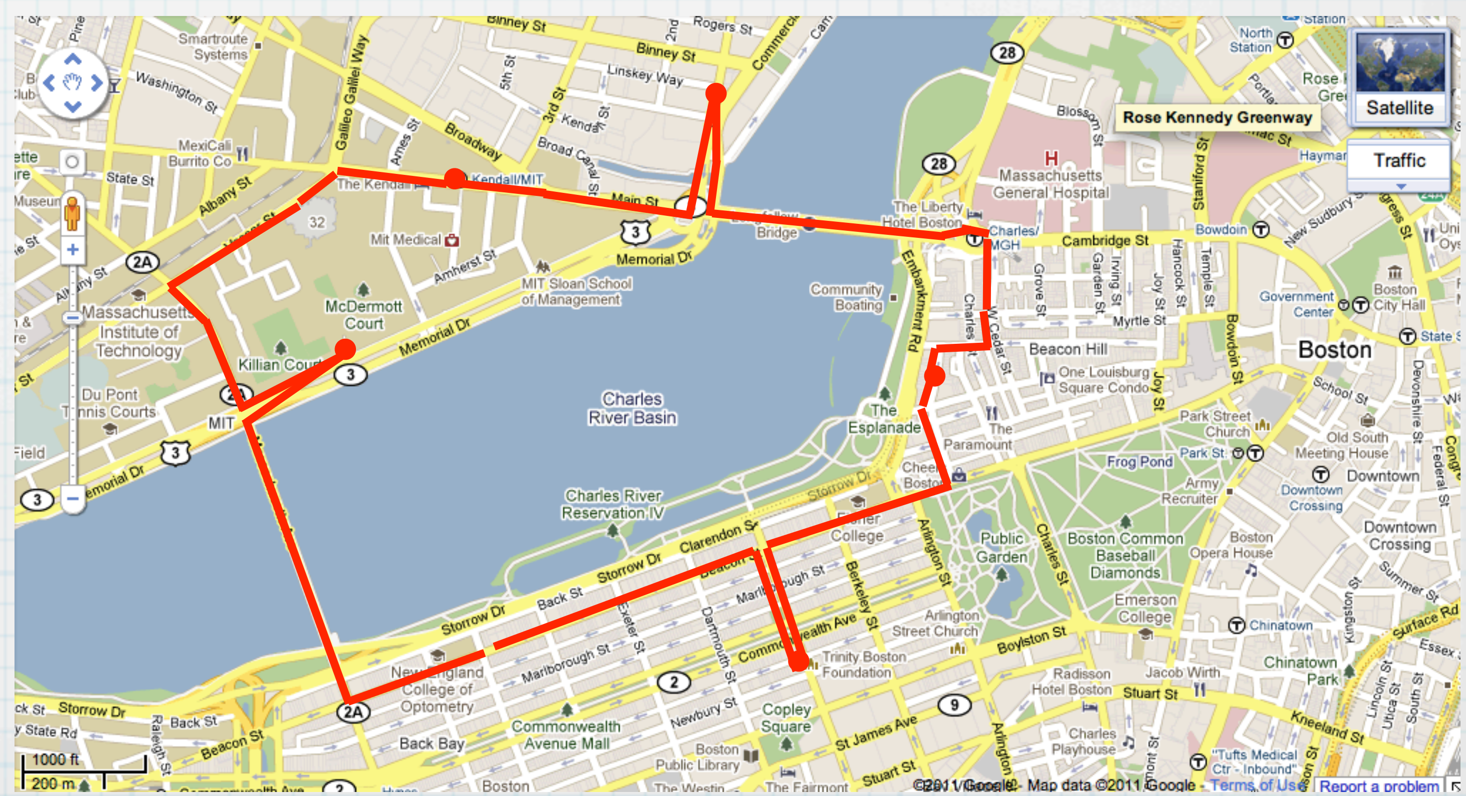
In framework for approximation scheme, choose $p = \varepsilon / 2(\varepsilon^{-1}+1)$ so increase in cost is at most ε OPT

Corollary: There is a linear-time approximation scheme for traveling salesman in planar graphs.

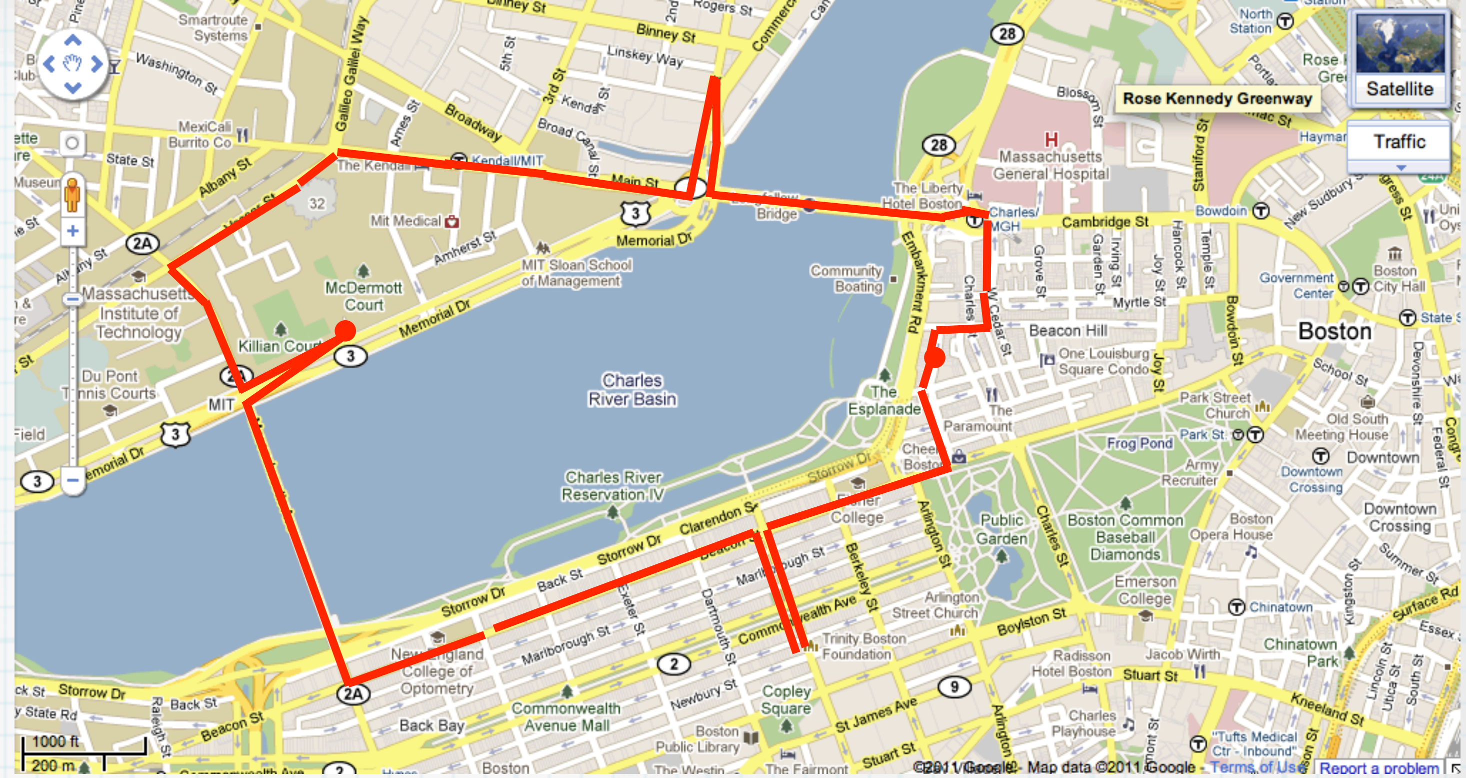
But we want to address...

Traveling salesman on a *subset* of vertices

Need a more general spanner result



Traveling salesman on a *subset* of vertices



We need a subgraph that approximately preserves distances between vertices of the subset.

Minimum weight to just preserve connectivity?

weight of minimum *Steiner tree* spanning the subset.

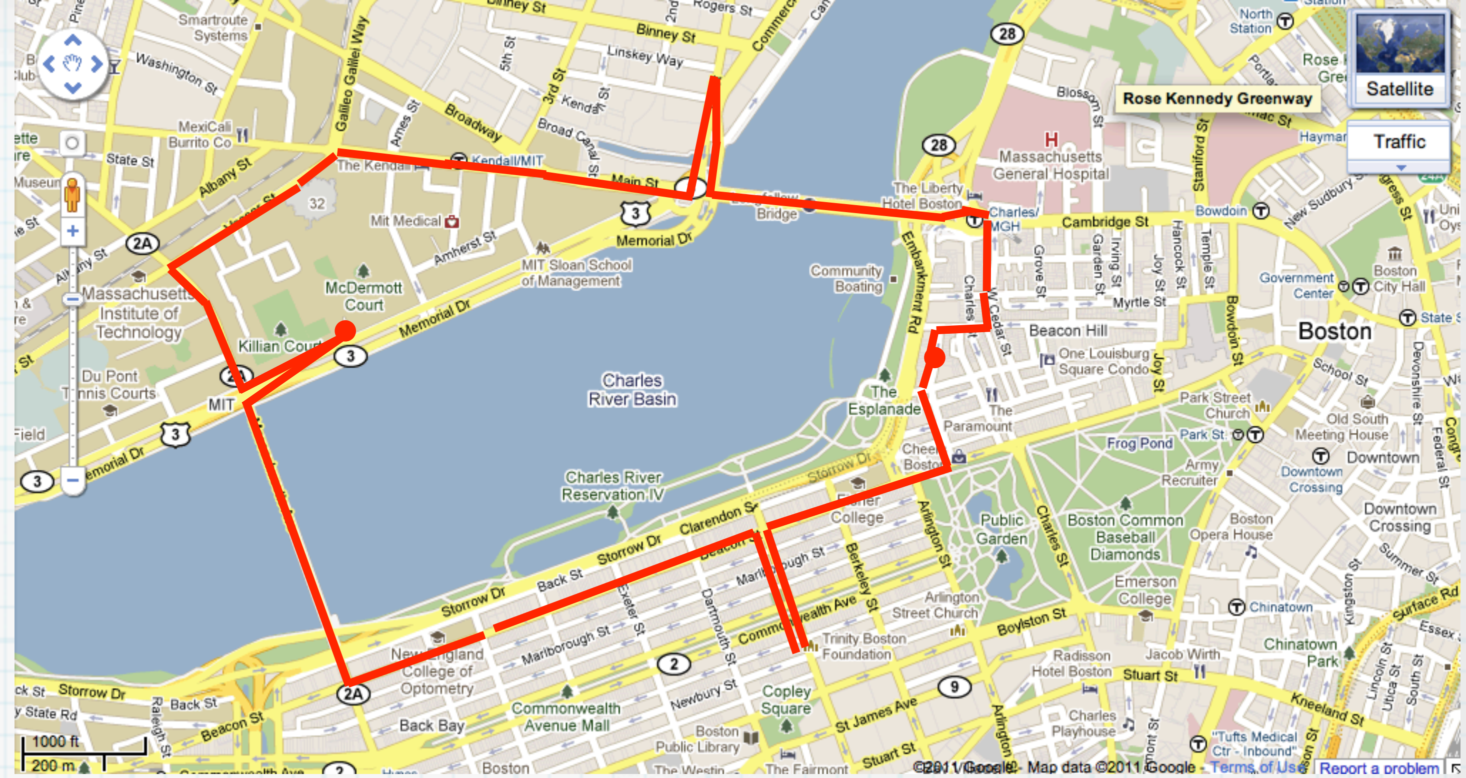
Theorem: for any undirected planar graph G with edge-weights, and any given subset S of vertices,

$\exists$ subgraph of weight $\leq f(\varepsilon)\times$ min Steiner tree weight such that,

$\forall u,v \in S$,

u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

Traveling salesman on a *subset* of vertices



We need a subgraph that approximately preserves distances between vertices of the subset.

Minimum weight to just preserve connectivity?

weight of minimum *Steiner tree* spanning the subset.

Theorem: for any undirected planar graph G with edge-weights, and any given subset S of vertices,

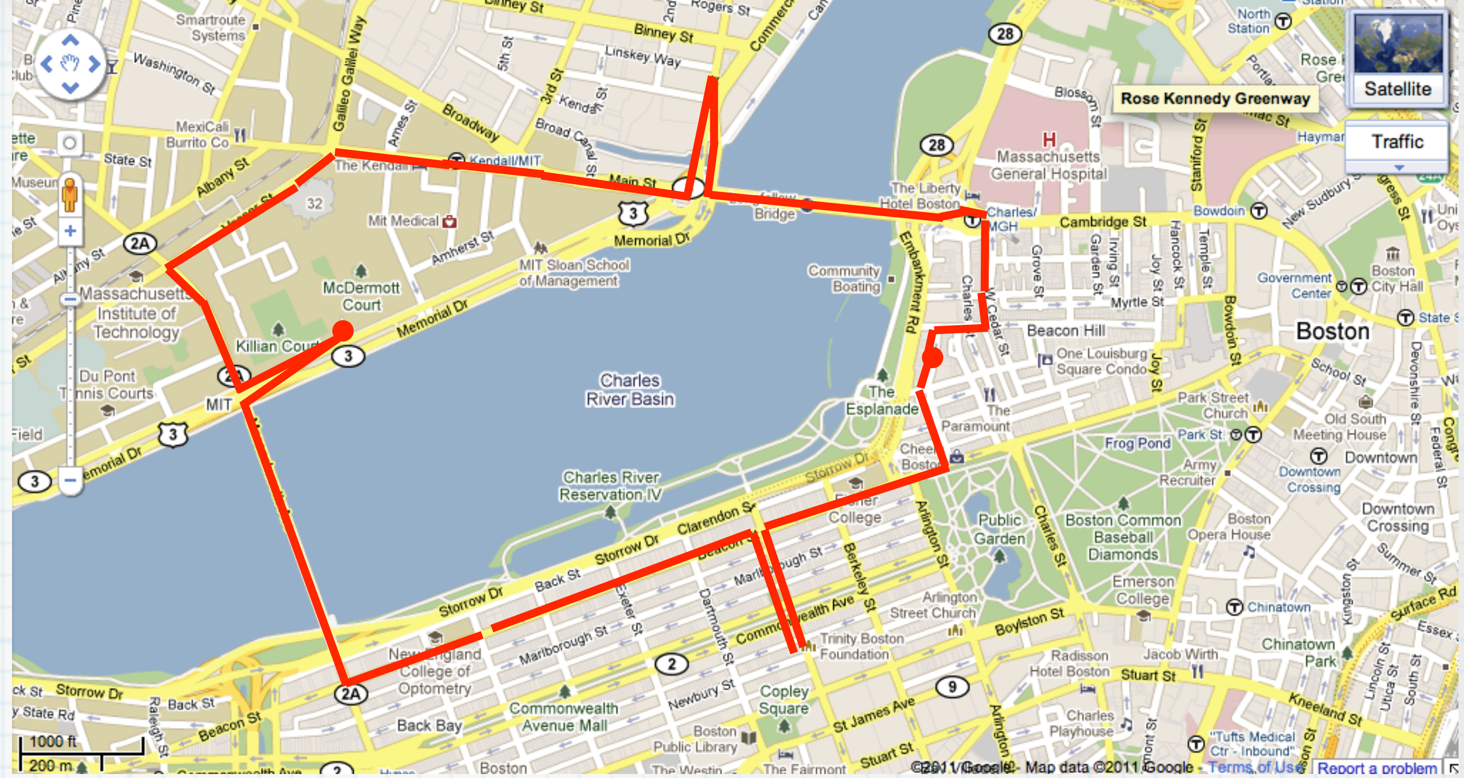
$\exists$ subgraph of weight $\leq f(\varepsilon)\times$ min Steiner tree weight such that,

$\forall u,v \in S$,

u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

Traveling salesman on a *subset* of vertices

We need a subgraph that approximately preserves distances between vertices of the subset.



Minimum weight to just preserve connectivity?

weight of minimum *Steiner tree* spanning the subset.

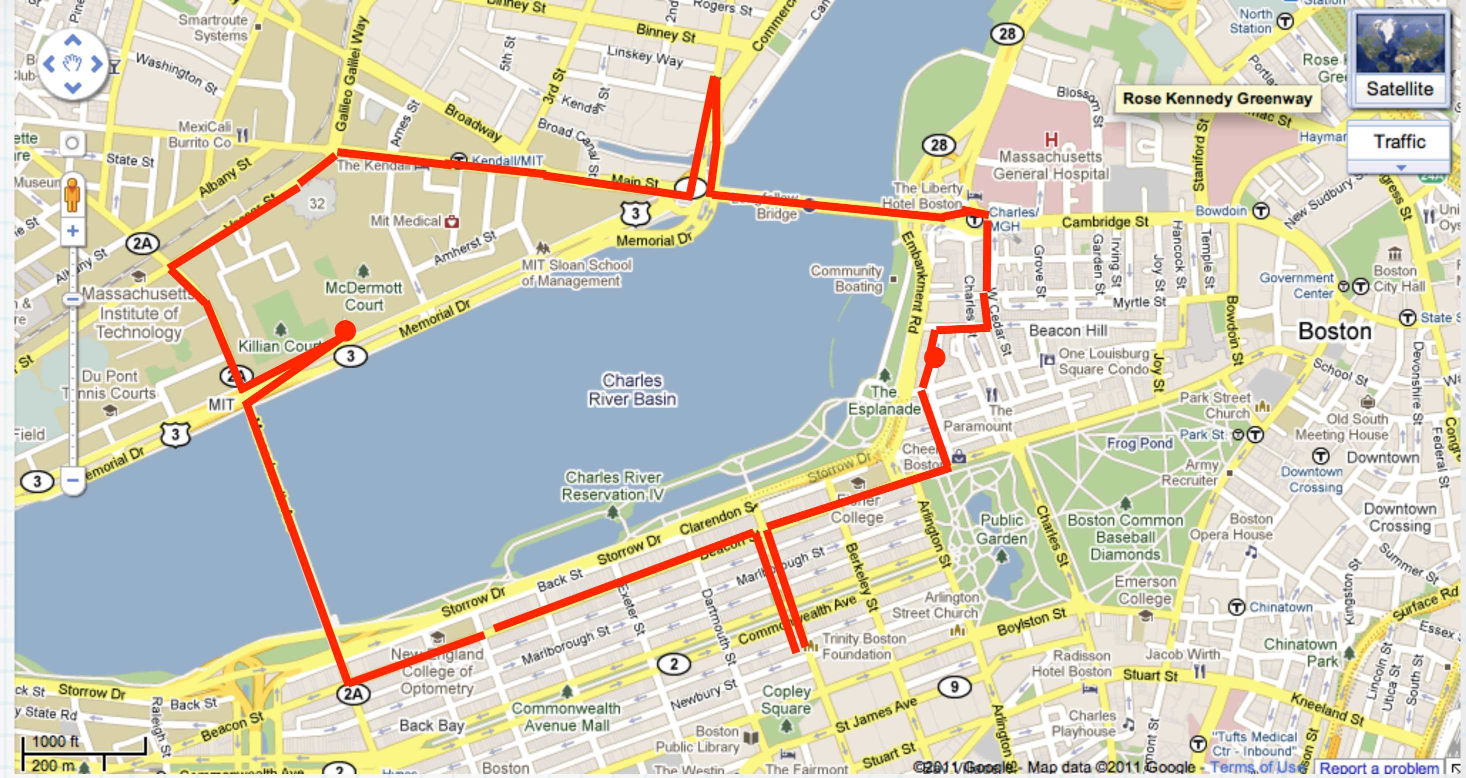
Theorem: for any undirected planar graph G with edge-weights, and any given subset S of vertices,

$\exists$ subgraph of weight $\leq f(\varepsilon)\times$ min Steiner tree weight such that,

$\forall u,v \in S$,

u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

Traveling salesman on a *subset* of vertices



We need a subgraph that approximately preserves distances between vertices of the subset.

Minimum weight to just preserve connectivity?

weight of minimum *Steiner tree* spanning the subset.

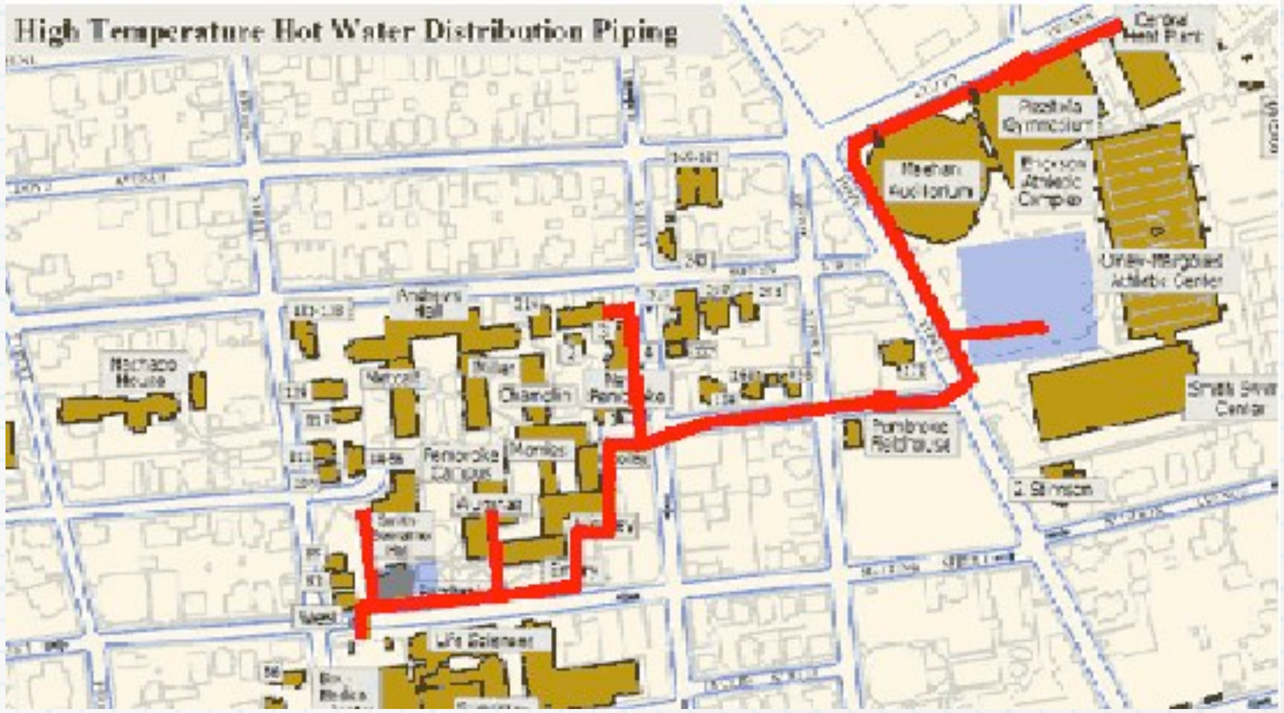
Theorem: for any undirected planar graph G with edge-weights, and any given subset S of vertices,

$\exists$ subgraph of weight $\leq f(\varepsilon) \times$ min Steiner tree weight such that,

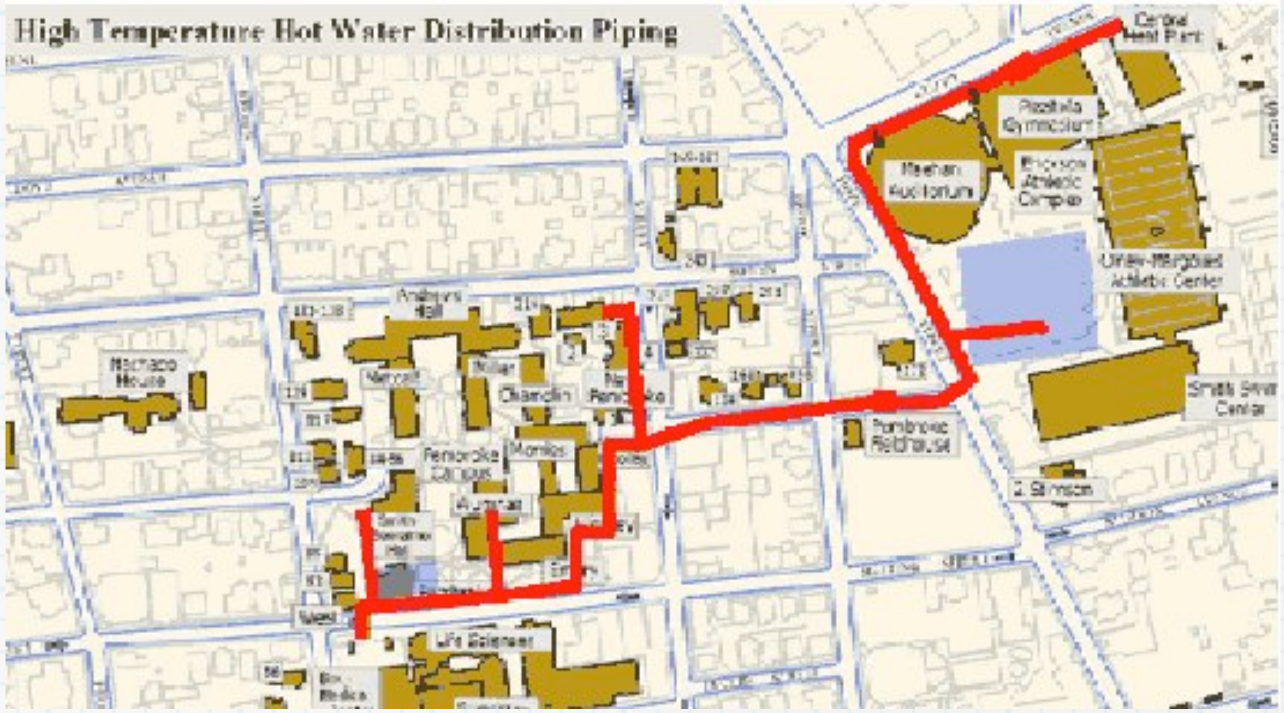
$\forall u, v \in S$,

u-to-v distance in subgraph $\leq (1+\varepsilon)$ u-to-v distance in G

Given subset S of vertices, we need a subgraph that approximately preserves Steiner tree weight.



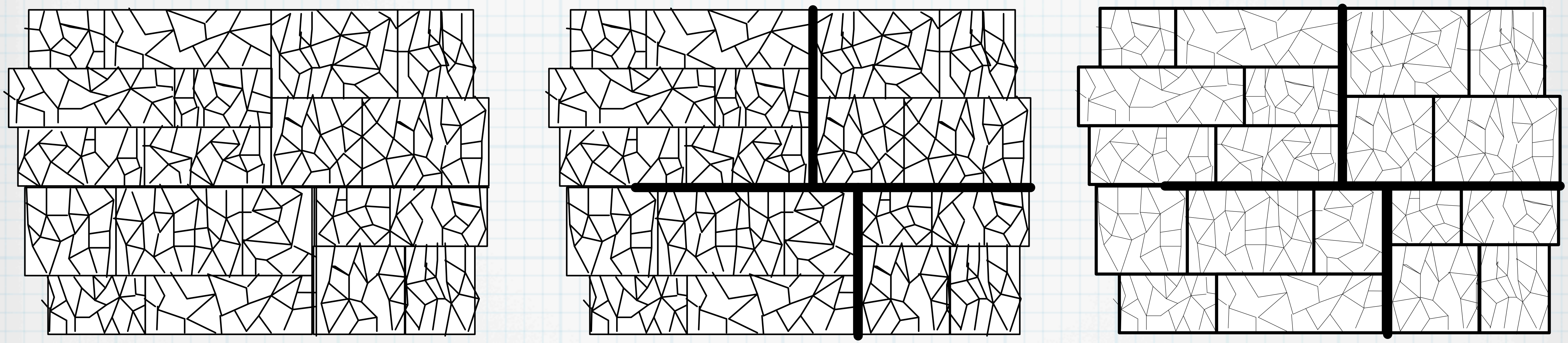
Steiner tree



Given subset S of vertices, we need a subgraph that approximately preserves Steiner tree weight.

Theorem: for any undirected planar graph G with edge-weights, and any given subset S of vertices,
$\exists$ subgraph of cost $\leq f(\varepsilon)\times$ min **Steiner** tree cost such that
min Steiner tree cost in subgraph $\leq (1+\varepsilon)$ min **Steiner tree** cost in G

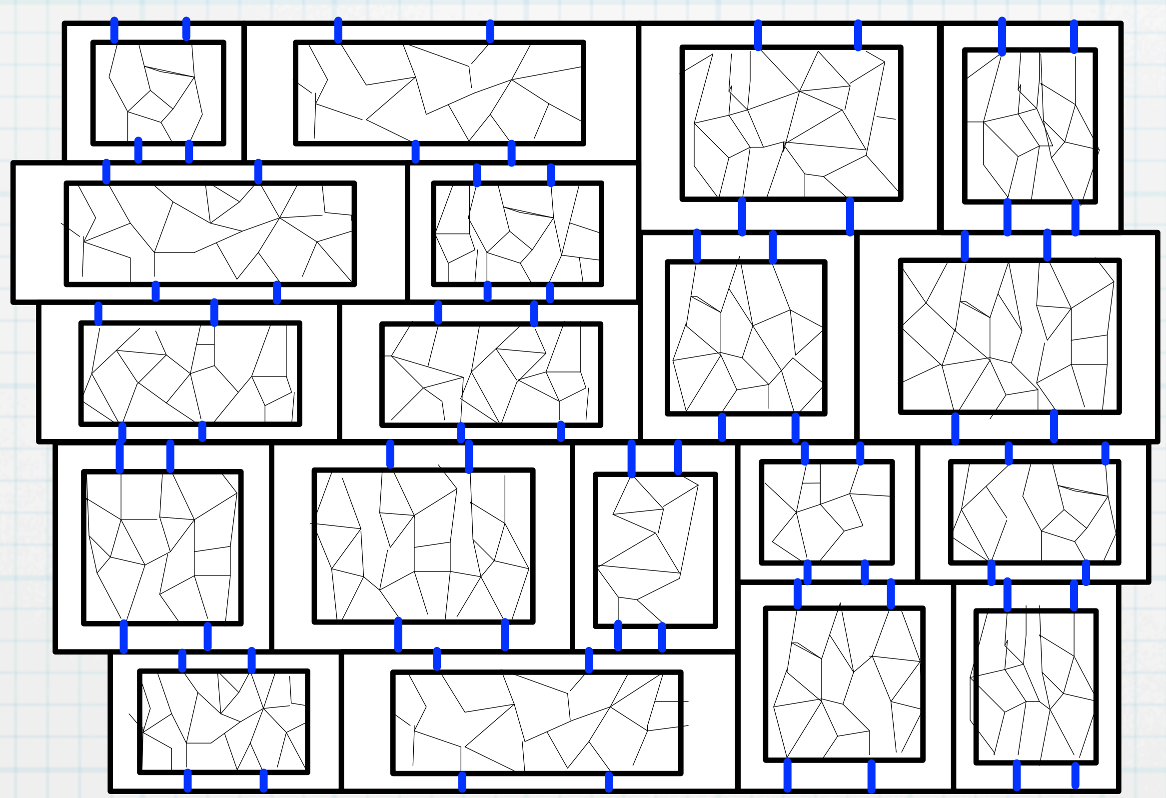
Tool for Step 1: *Brick Decomposition*

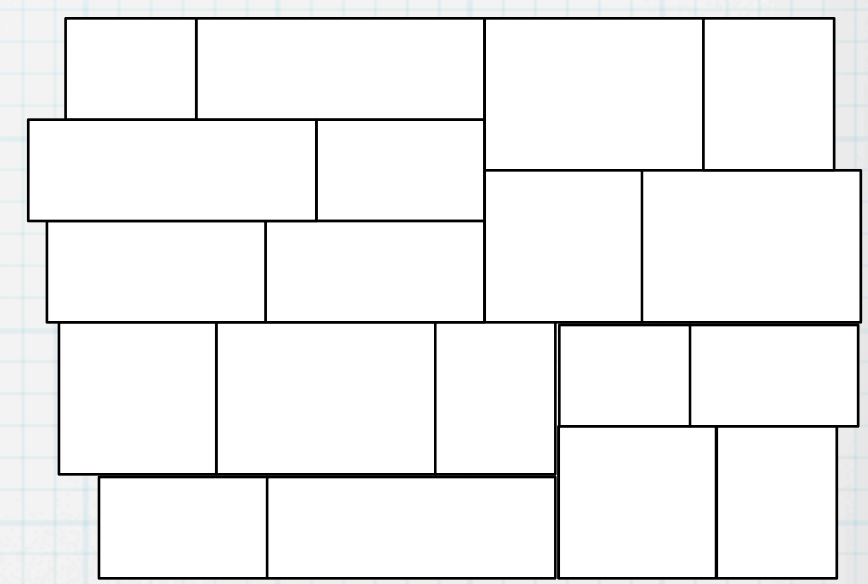


T:= 2-approx. Steiner tree

M := subgraph containing T

Bricks := faces of M

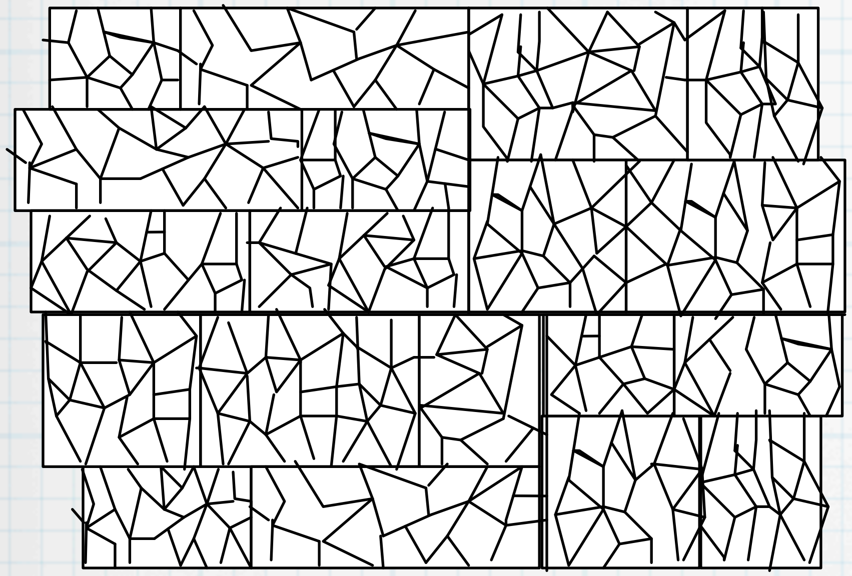




Connect each brick to copy of M using $c(\varepsilon)$ portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

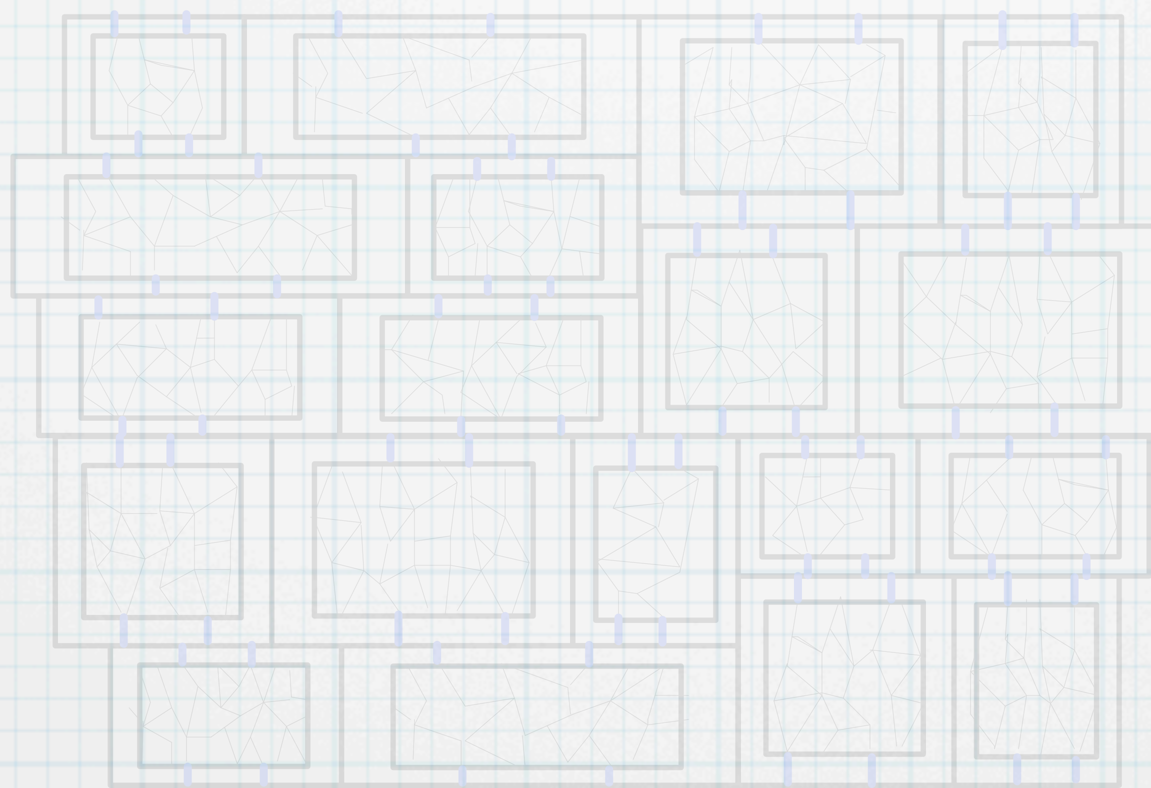
Tool for Step 1: *Brick Decomposition*

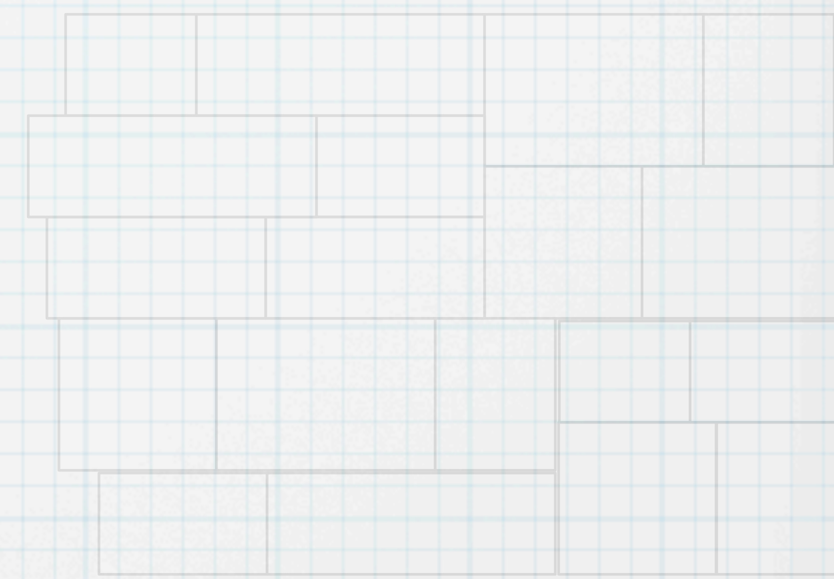


T:= 2-approx. Steiner tree

M := subgraph containing *T*

Bricks := faces of *M*

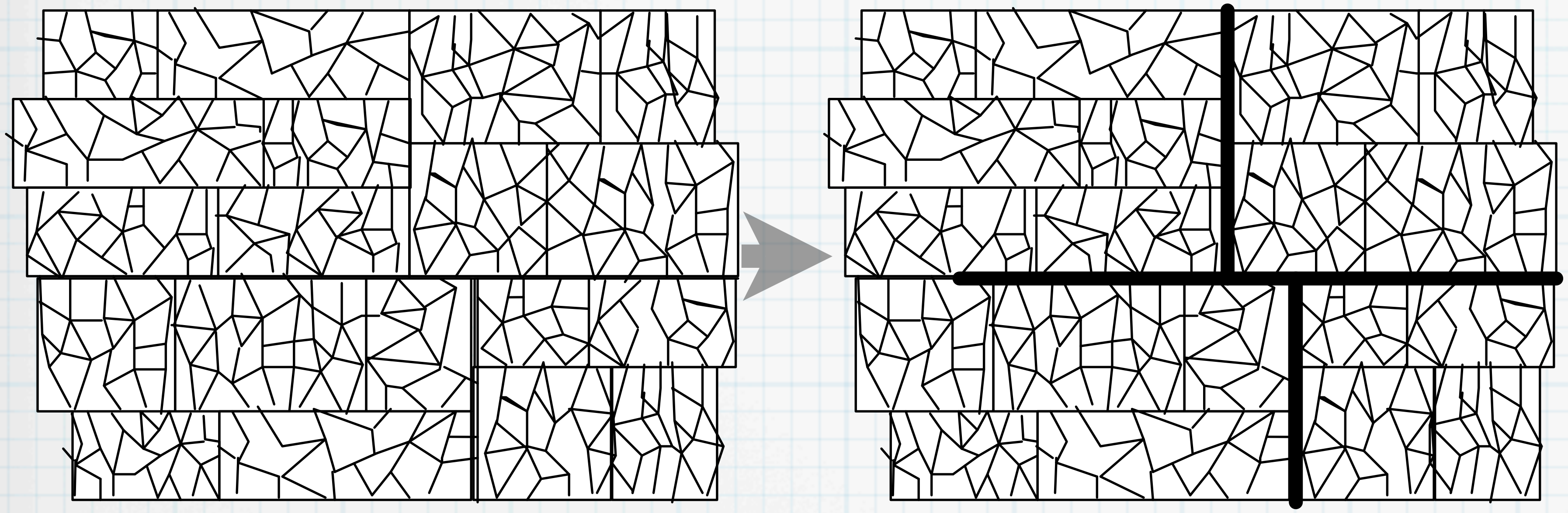




Connect each brick to copy of *M* using *c(ε)* portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

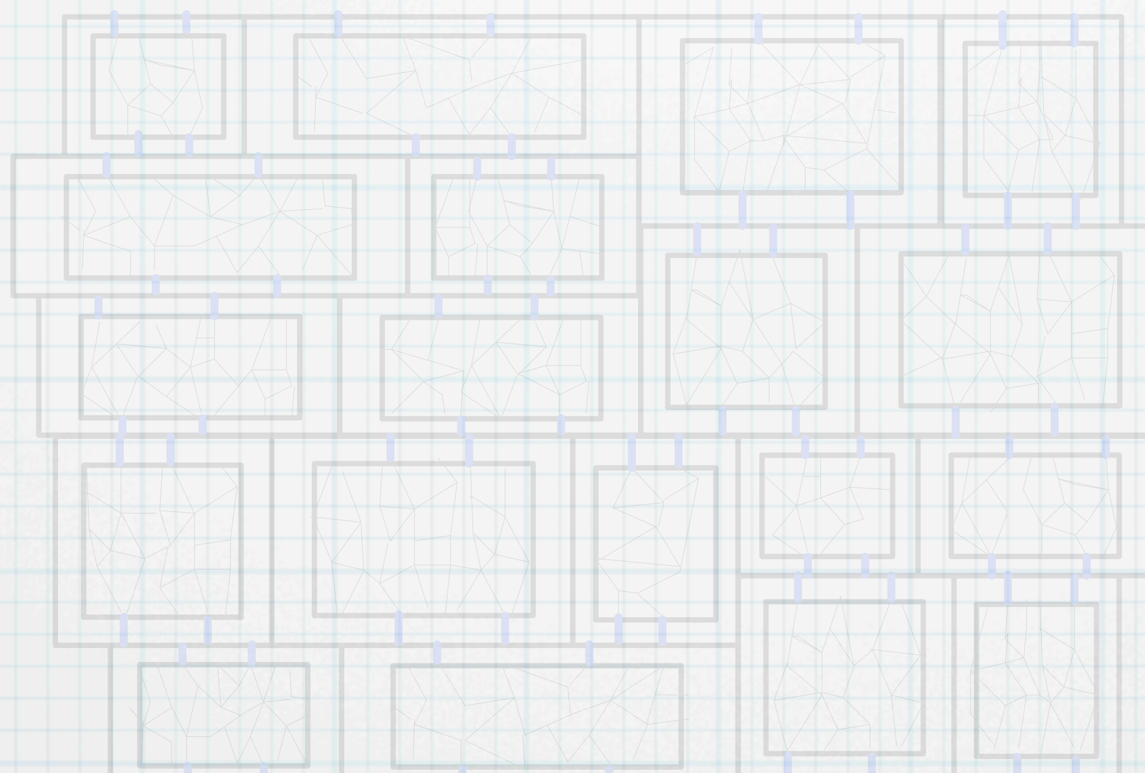
Tool for Step 1: *Brick Decomposition*



T:= 2-approx. Steiner tree

M := subgraph containing *T*

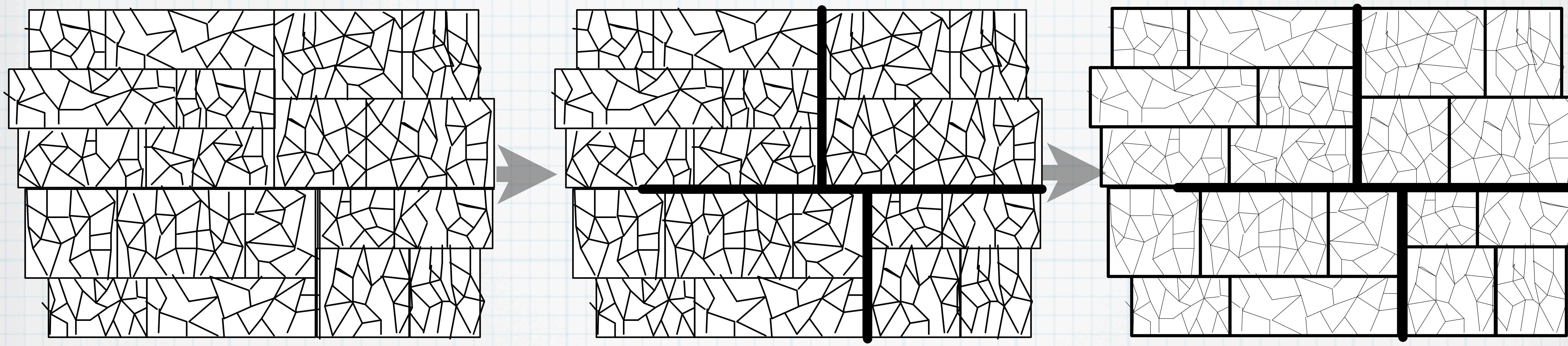
Bricks := faces of *M*



Connect each brick to copy of *M* using *c(ε)* portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

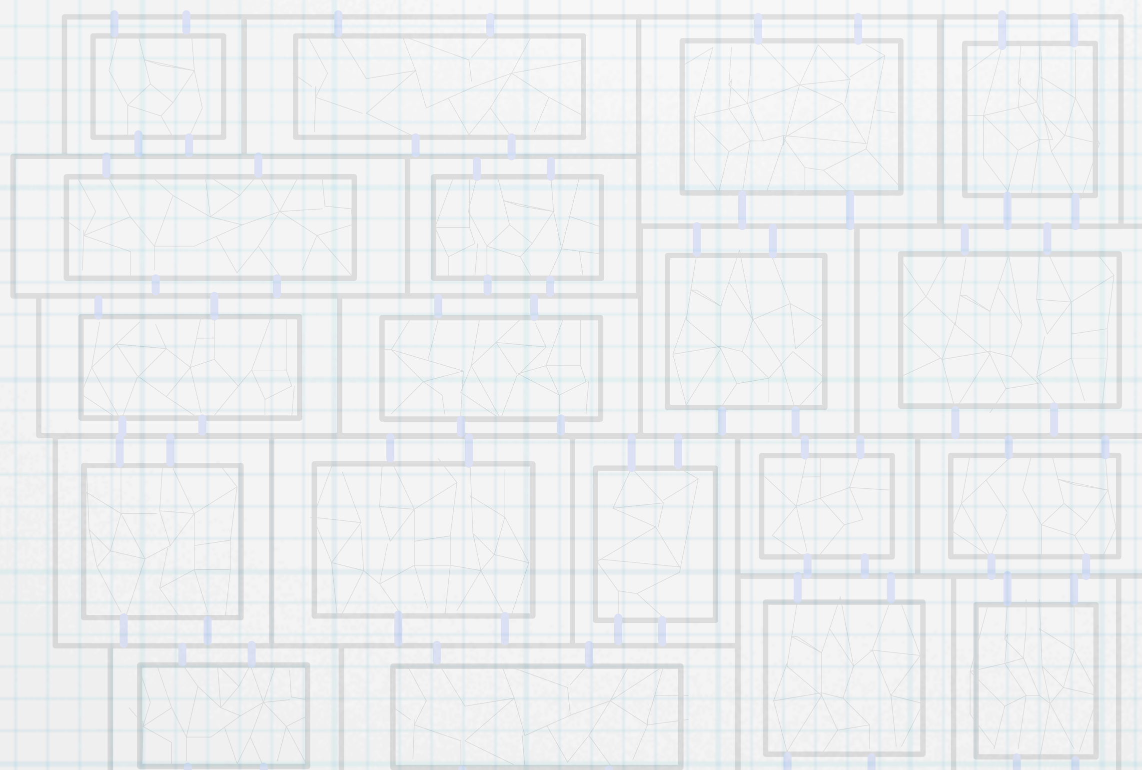
Tool for Step 1: *Brick Decomposition*

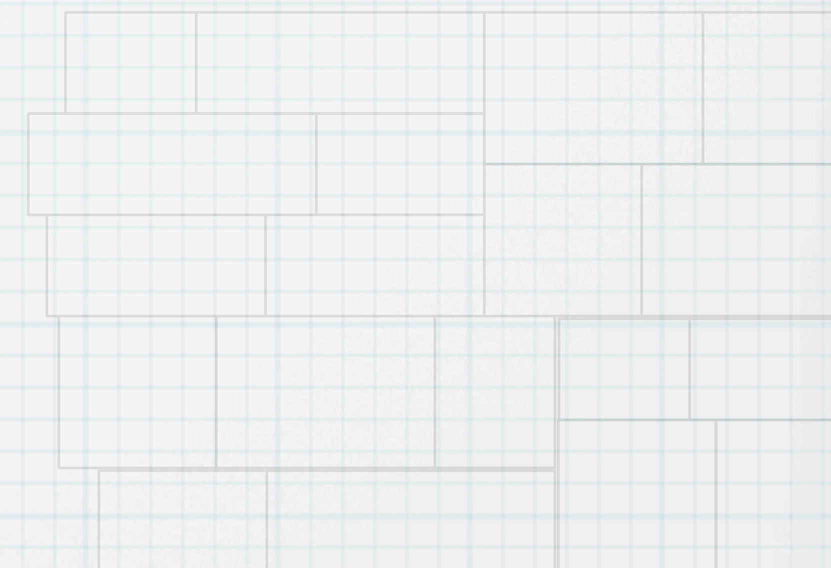


T:= 2-approx. Steiner tree

M := subgraph containing T

Bricks := faces of M

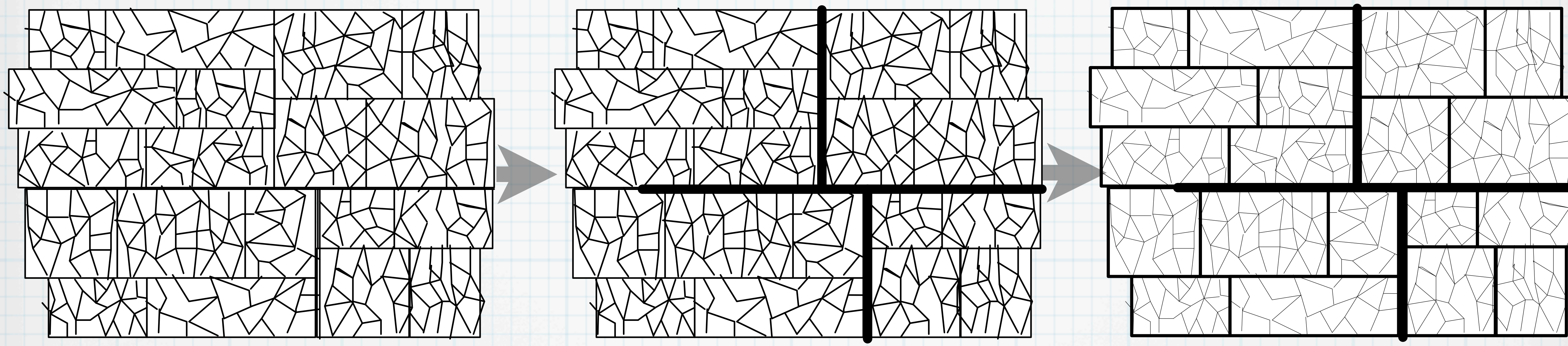


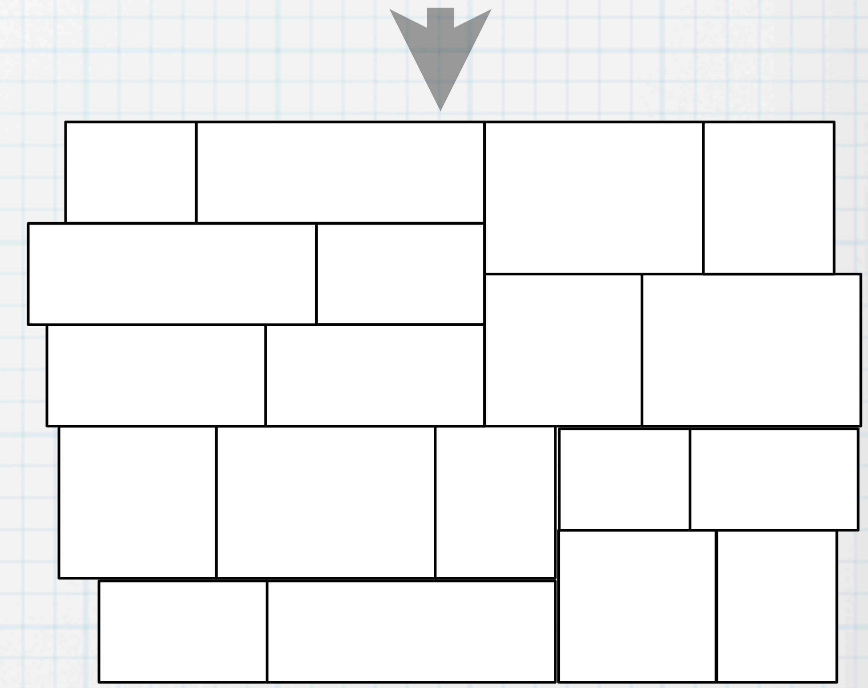


Connect each brick to copy of M using $c(\varepsilon)$ portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

Tool for Step 1: *Brick Decomposition*

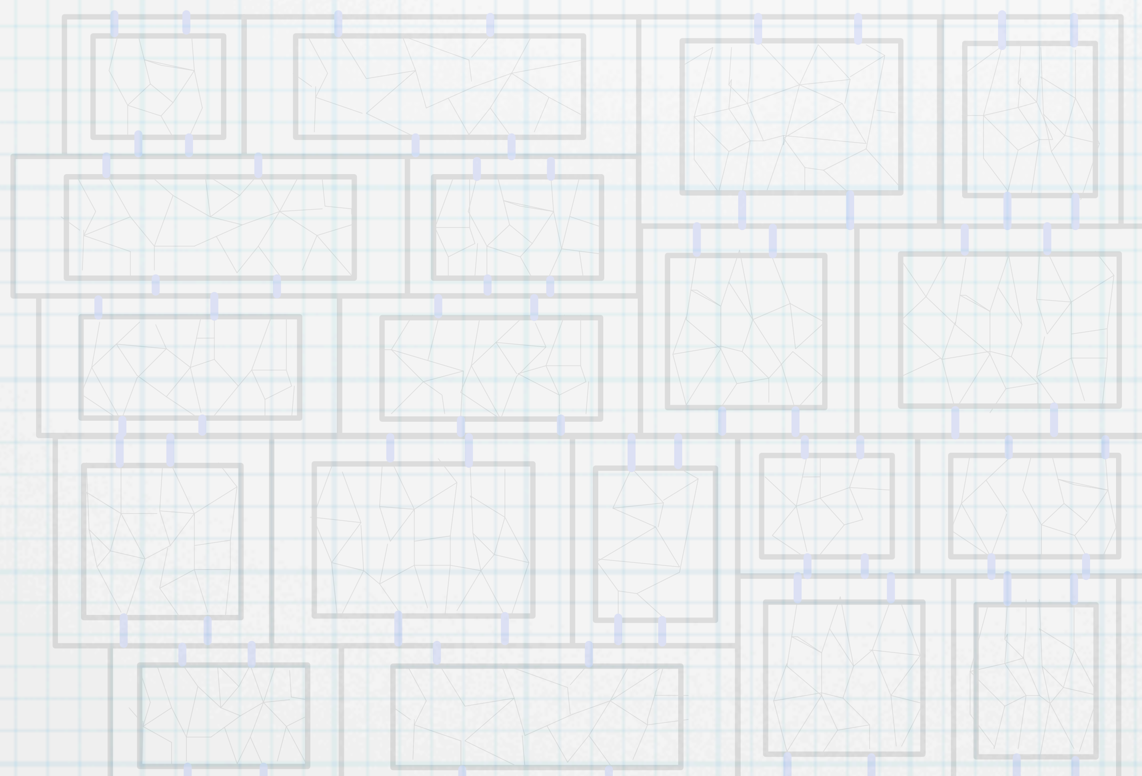




T:= 2-approx. Steiner tree

M := subgraph containing *T*

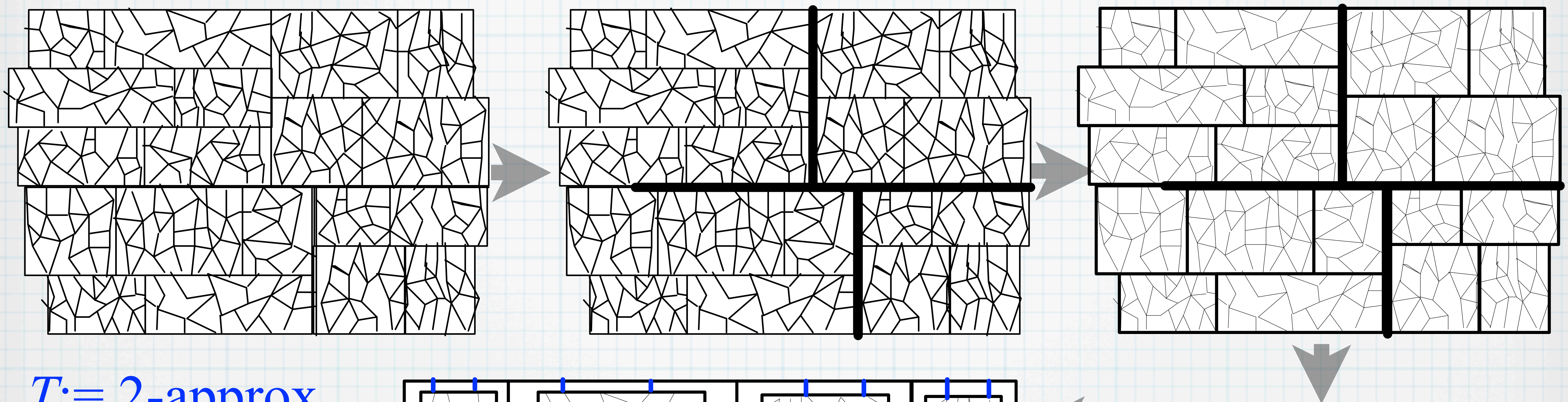
Bricks := faces of *M*



Connect each brick to copy of *M* using *c(ε)* portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

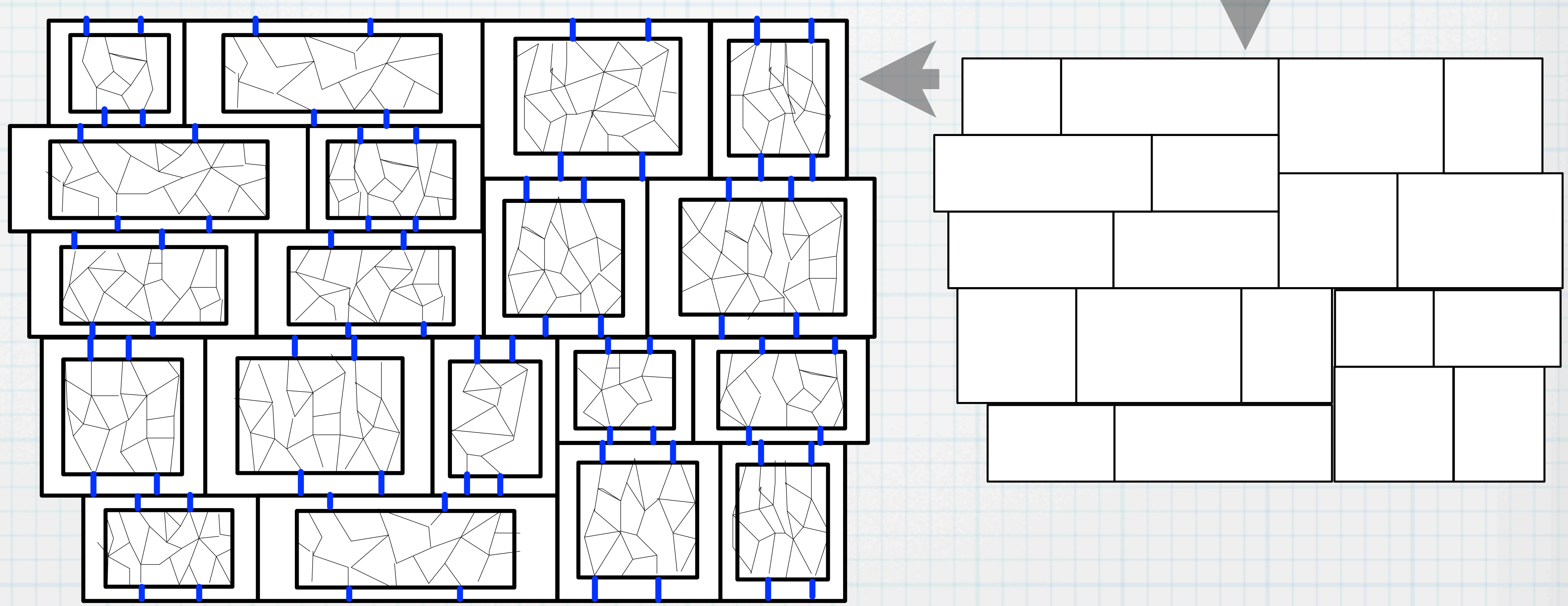
Tool for Step 1: *Brick Decomposition*



T:= 2-approx. Steiner tree

M := subgraph containing *T*

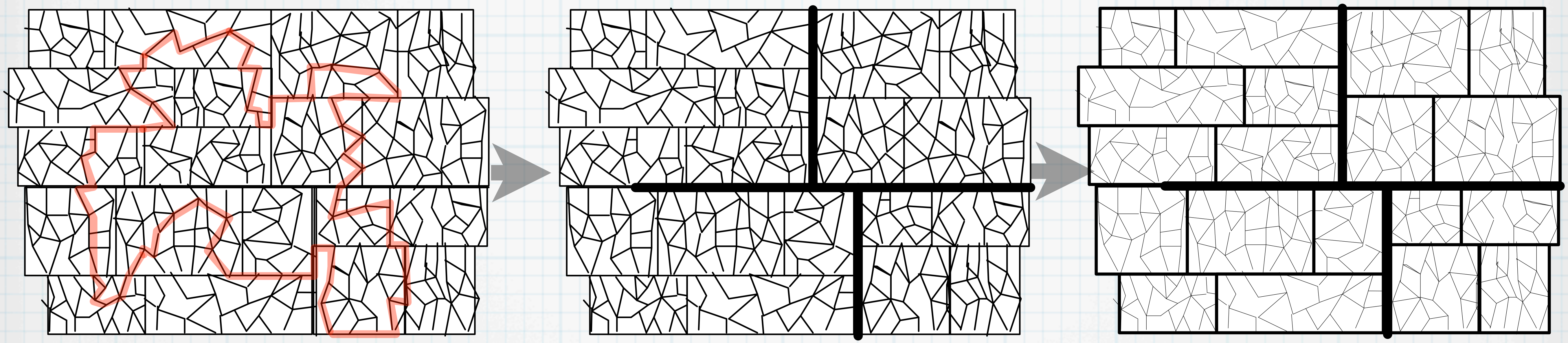
Bricks := faces of *M*

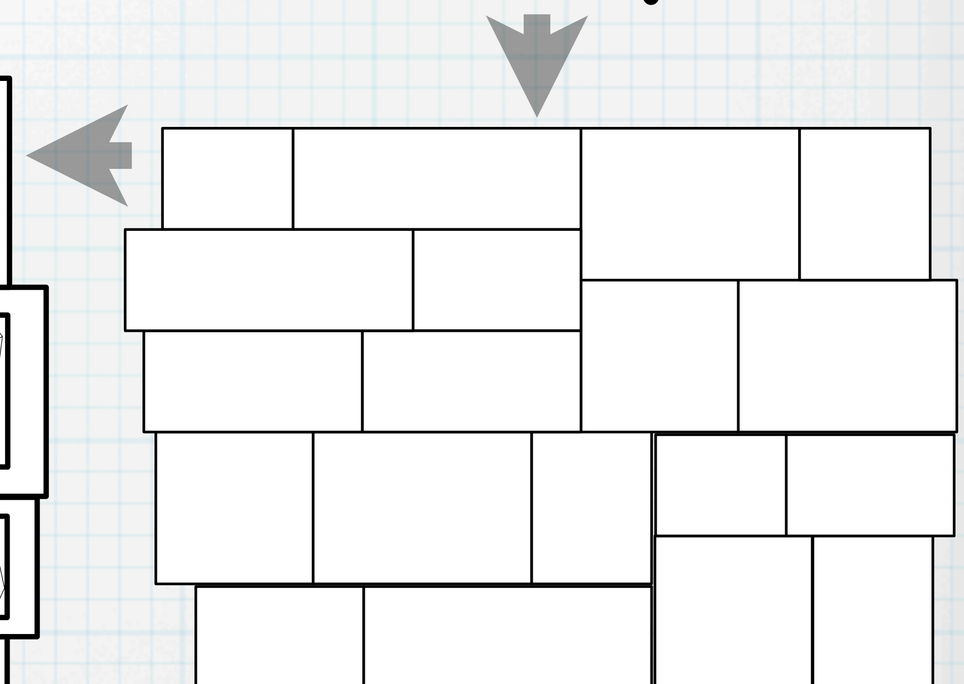


Connect each brick to copy of *M* using *c(ε)* portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

Tool for Step 1: *Brick Decomposition*

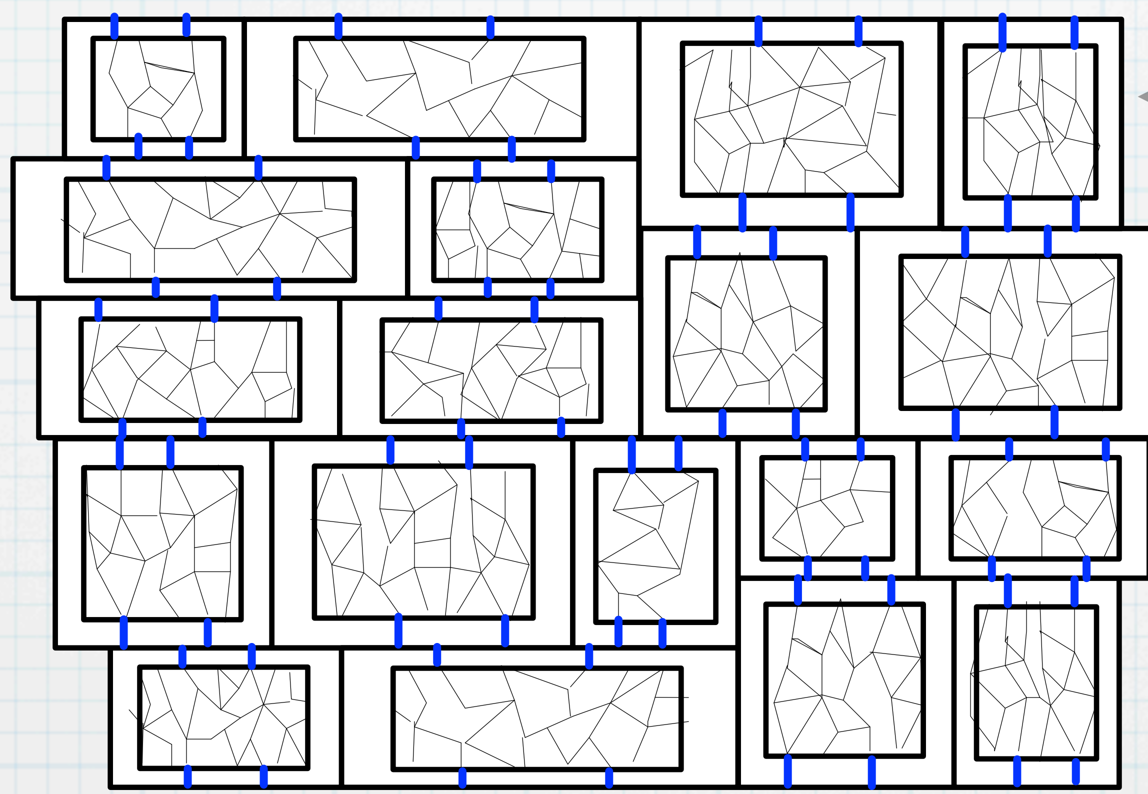




T:= 2-approx. Steiner tree

M := subgraph containing T

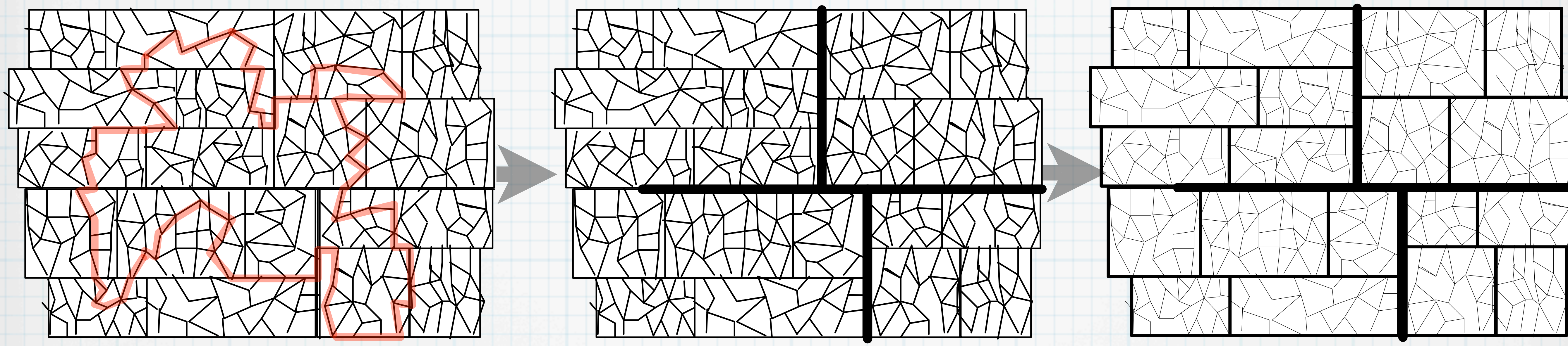
Bricks := faces of M



Connect each brick to copy of M using $c(\varepsilon)$ portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

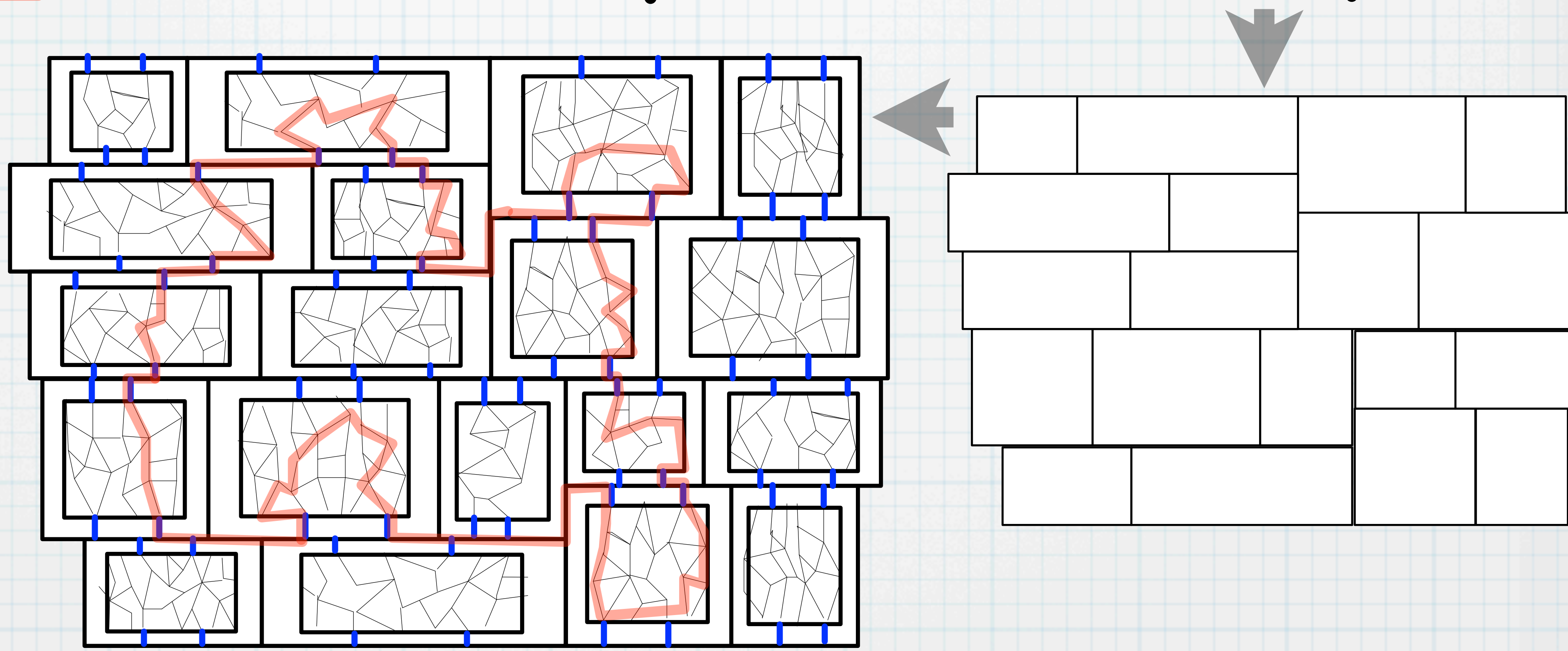
Tool for Step 1: *Brick Decomposition*



T:= 2-approx. Steiner tree

M := subgraph containing T

Bricks := faces of M

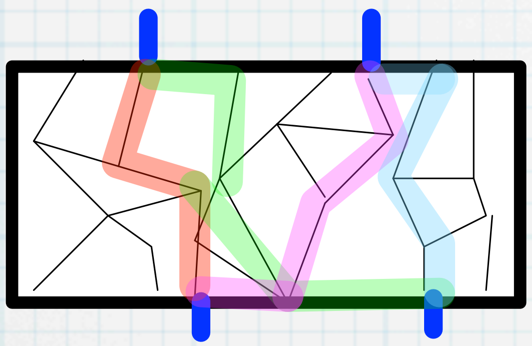


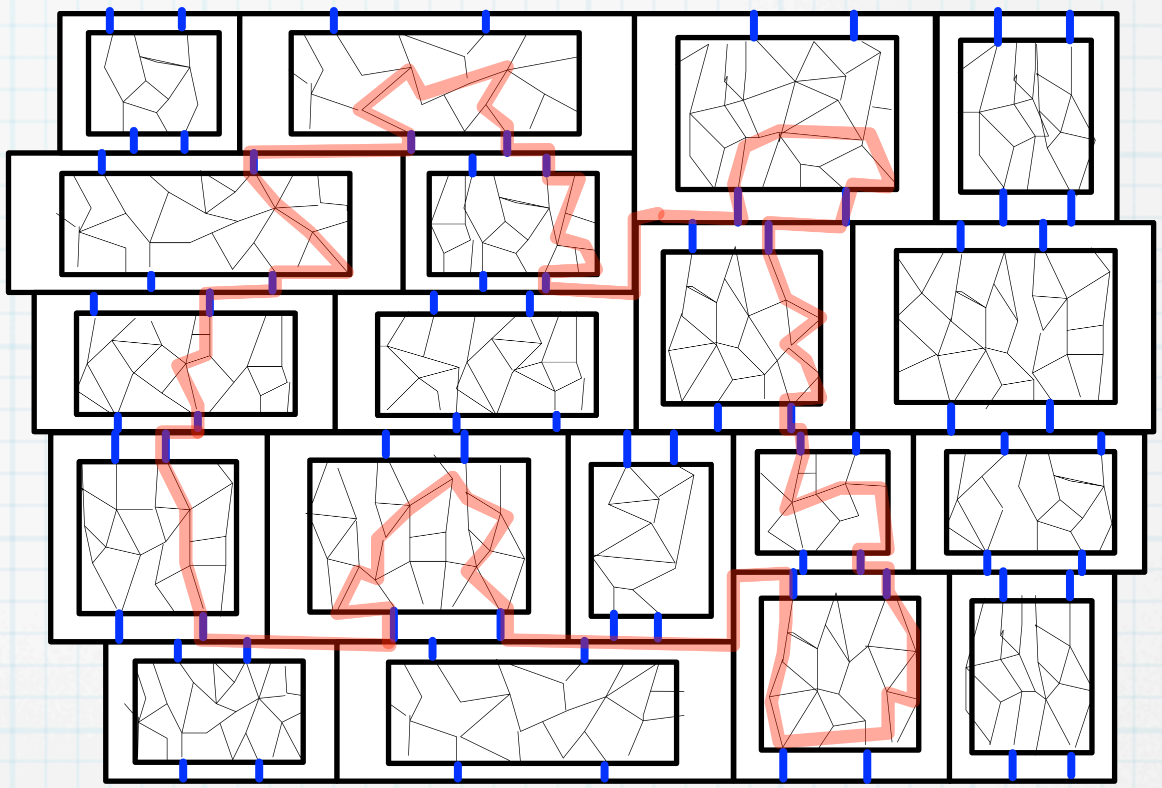
Connect each brick to copy of M using $c(\varepsilon)$ portal edges

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

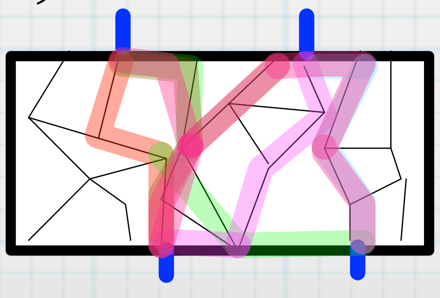
TSP Spanner construction: Include M and, for each brick B, all portal-to-portal shortest-paths within B.





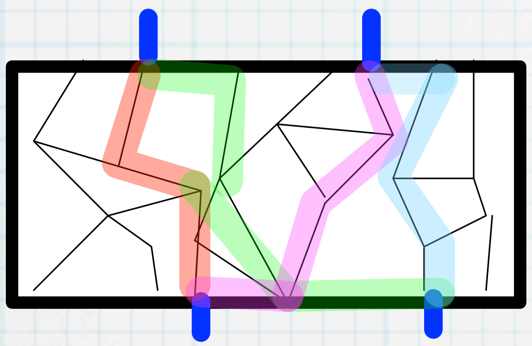
Steiner Structure Theorem: There is a 1+ ε-approx. Steiner tree that uses portal edges to go between bricks.

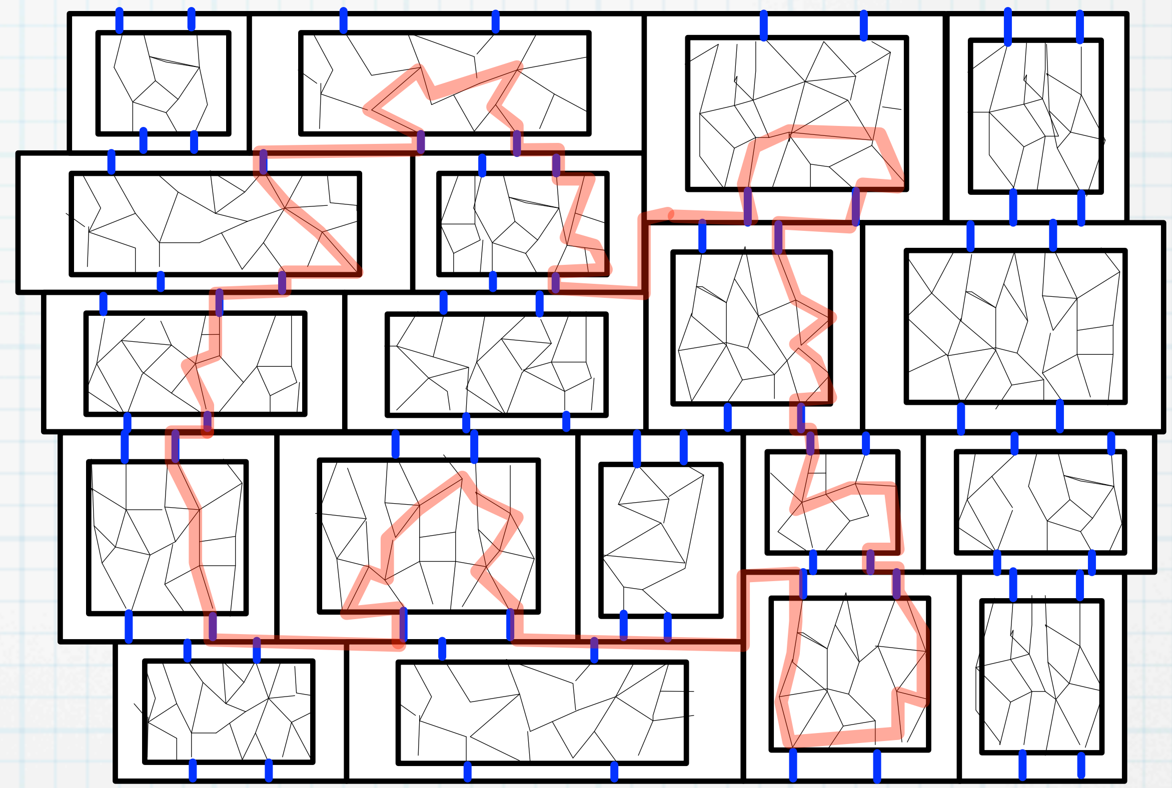
Steiner Spanner construction: Include M and, for each brick B, for each subset of portal ends, the min Steiner tree within B.



TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

TSP Spanner construction: Include M and, for each brick B, all portal-to-portal shortest-paths within B.



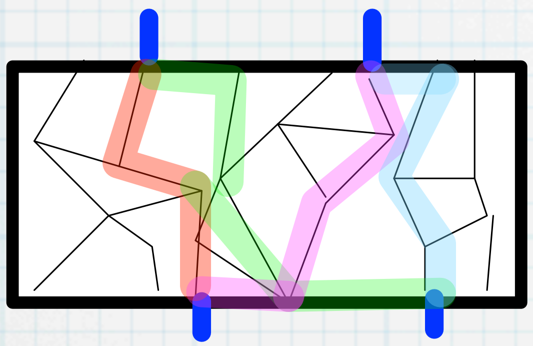


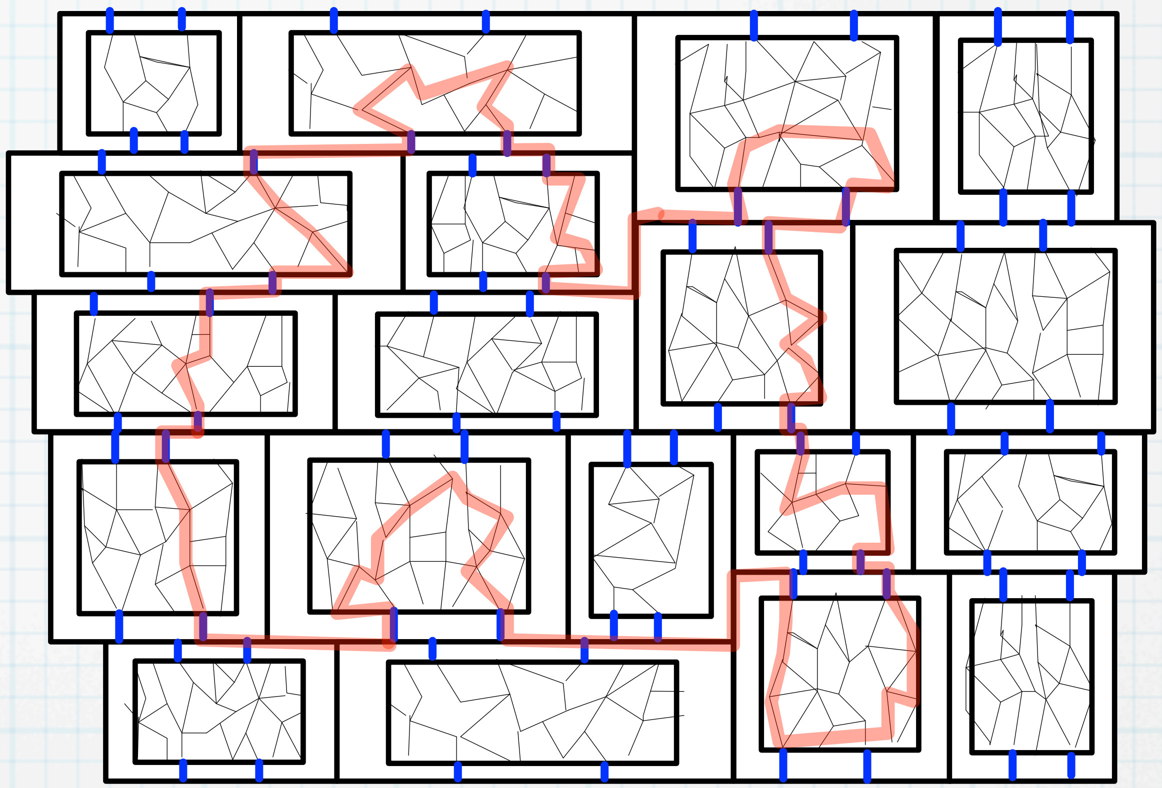
Steiner Structure Theorem: There is a 1+ ε-approx. Steiner tree that uses portal edges to go between bricks.

Steiner Spanner construction: Include M and, for each brick B, for each subset of portal ends, the min Steiner tree within B.

TSP Structure Theorem: There is a 1+ ε-approx. tour that uses portal edges to go between bricks.

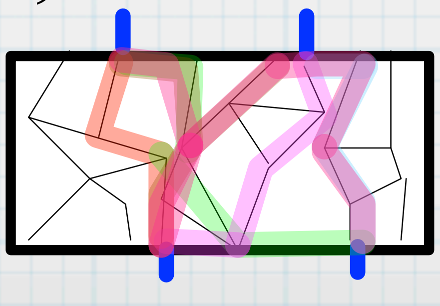
TSP Spanner construction: Include M and, for each brick B, all portal-to-portal shortest-paths within B.





Steiner Structure Theorem: There is a 1+ ε-approx. Steiner tree that uses portal edges to go between bricks.

Steiner Spanner construction: Include M and, for each brick B, for each subset of portal ends, the min Steiner tree within B.



Remarks about spanner methodology

- Brick-decomposition construction takes *O(n log n)* time.

- There's a way to use brick decompositions that avoid some of the overhead of the spanner methodology, leads to better dependence on ε.

Steiner-tree approximation scheme has been implemented!
(Constants tweaked to get a fast algorithm that gets very good solutions.)
[Tazari, Müller-Hannemann, 2009]

- Brick decomposition can start with any *connected* subgraph, not just tree.

- To cope with disconnected subgraphs, *prize-collecting clustering* (MohammadTaghi's talk) has become an essential technique.

Open problems

Lots! We're just gaining steam.

Will need new techniques for these problems....

- Facility location problems
- Vehicle routing problems
- *k*-tree
- vertex-weighted Steiner tree
- directed Steiner tree
- two-edge connected Steiner
- two-vertex-connected Steiner

Open problems

Lots! We're just gaining steam.

Will need new techniques for these problems....

- Facility location problems
- Vehicle routing problems
- *k*-tree
- vertex-weighted Steiner tree
- directed Steiner tree
- two-edge connected Steiner
- two-vertex-connected Steiner

Advertisements:

I'm writing a book about (some) optimization algorithms for planar graphs. Email me if you want to receive a draft.

Also, we are working to develop a library of reference implementations of planar-graph algorithms.