A subexponential parameterized algorithm for Subset TSP on planar graphs

Philip N. Klein[1] Dániel Marx[2]

[1]Brown University,
Providence, Rhode Island, USA

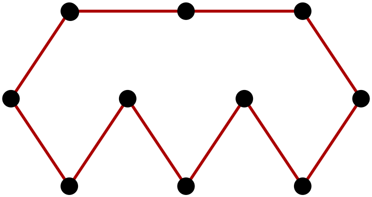
[2]Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

Dagstuhl Seminar 13331
August 15, 2013

TSP

TSP

Input: A set T of cities and a distance function d on T
Output: A tour on T with minimum total distance

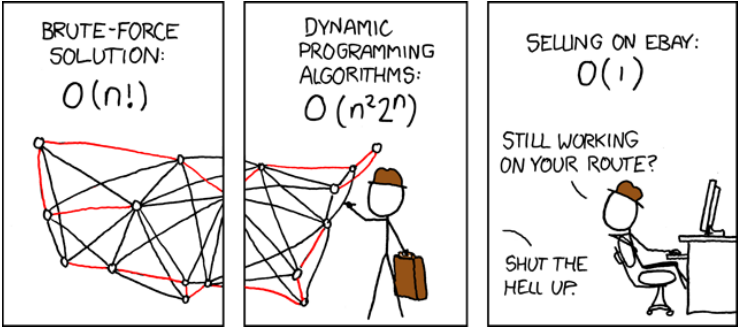


Theorem [Held and Karp 1962]

TSP with n cities can be solved in time $2^n \cdot n^2 \cdot \log D$, where D is the maximum (integer) distance.

Dynamic programming:
Let $x(v, T')$ be the minimum length of path from v_{start} to v visiting all the cities $T' \subseteq T$.

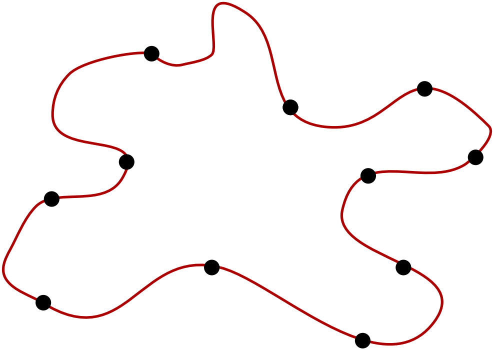
Obligatory cartoon



http://xkcd.com/399/

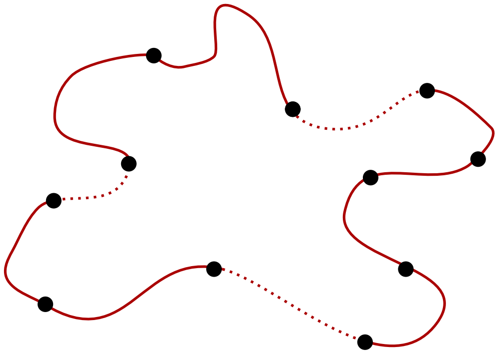
c-change TSP

- c-change operation: removing c steps of the tour and connecting the resulting c paths in some other way.
- A solution is c-OPT if no c-change can improve it.
- We can find a c-OPT solution in $n^{O(c)} \cdot D$ time, where D is the maximum (integer) distance.



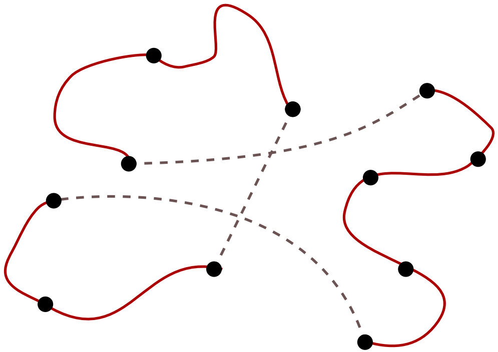
c-change TSP

- c-change operation: removing c steps of the tour and connecting the resulting c paths in some other way.
- A solution is c-OPT if no c-change can improve it.
- We can find a c-OPT solution in $n^{O(c)} \cdot D$ time, where D is the maximum (integer) distance.



c-change TSP

- c-change operation: removing c steps of the tour and connecting the resulting c paths in some other way.
- A solution is c-OPT if no c-change can improve it.
- We can find a c-OPT solution in $n^{O(c)} \cdot D$ time, where D is the maximum (integer) distance.



c-change TSP

- Finding a 2-OPT or 3-OPT tour is a popular starting point for heuristics.
- Supposedly, finding a k-OPT tour for larger k is better (less likely to get stuck in a local optimum), but more time consuming.

c-change TSP

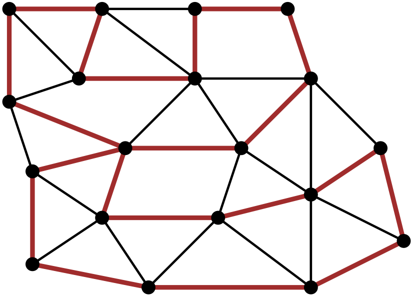
- Finding a 2-OPT or 3-OPT tour is a popular starting point for heuristics.
- Supposedly, finding a k-OPT tour for larger k is better (less likely to get stuck in a local optimum), but more time consuming.
- Unlikely that there is a fast algorithm for finding a k-OPT tour:

Theorem [M. 2008]

Finding a better tour in the k-change neighborhood is W[1]-hard.

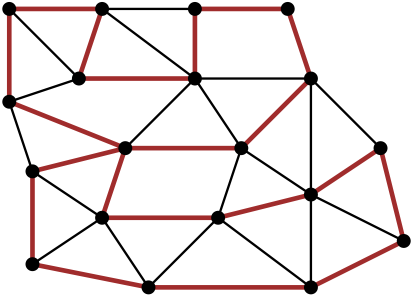
TSP on planar graphs

Assume that the cities correspond to the set of all vertices of a (weighted) planar graph and distance is measured in this (weighted) planar graph.



TSP on planar graphs

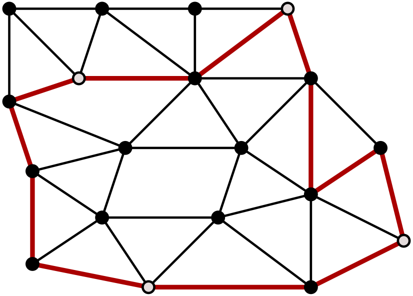
Assume that the cities correspond to the set of all vertices of a (weighted) planar graph and distance is measured in this (weighted) planar graph.



- Can be solved in time $n^{O(\sqrt{n})}$.
- Admits a PTAS.

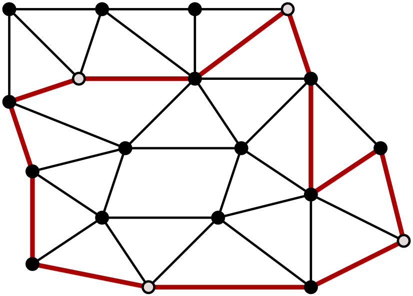
Subset TSP on planar graphs

Assume that the cities correspond to a subset T of vertices of a planar graph and distance is measured in this planar graph.



Subset TSP on planar graphs

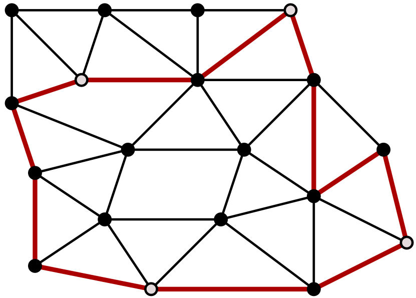
Assume that the cities correspond to a subset T of vertices of a planar graph and distance is measured in this planar graph.



- Can be solved in time $n^{O(\sqrt{n})}$.
- Can be solved in time $2^k \cdot n^{O(1)}$.
- **Question:** Can we solve it in time $2^{O(\sqrt{k})} \cdot n^{O(1)}$?

Subset TSP on planar graphs

Assume that the cities correspond to a subset T of vertices of a planar graph and distance is measured in this planar graph.

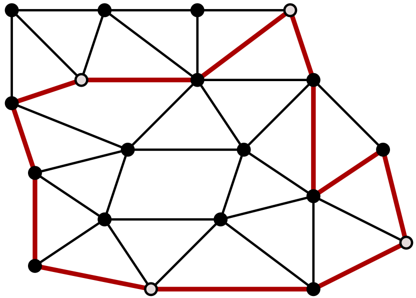


Theorem

Subset TSP for k cities in a unit-weight planar graph can be solved in time $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$.

Subset TSP on planar graphs

Assume that the cities correspond to a subset T of vertices of a planar graph and distance is measured in this planar graph.



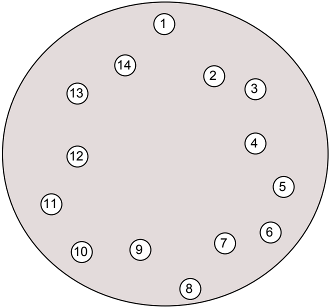
Theorem

Subset TSP for k cities in a weighted planar graph can be solved in time $(2^{O(\sqrt{k}\log k)} + W) \cdot n^{O(1)}$ if the weights are integers not more than W.

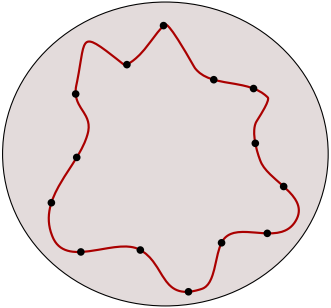
Two interpretations

Two possible interpretations for a solution of SUBSET TSP:

a cyclic ordering of the cities



closed walk in the graph

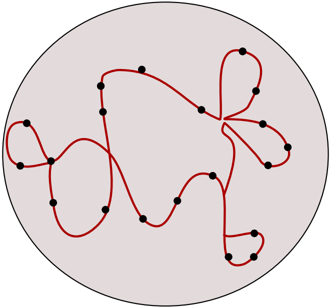


We can get the second from the first by concatenating shortest paths between adjacent cities in the ordering.

Technicalities

The closed walk can be degenerate in several ways:

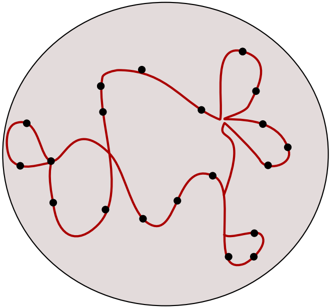
- can touch itself,
- can cross itself,
- can use an edge up to twice,
- can visit a city more than once.



We mostly ignore these technicalities in this talk.

Non-self-crossing

Definition: **Non-self-crossing** closed walk.
Definition: A tour is **non-self-crossing** if there is a non-self-crossing closed walk realizing it.

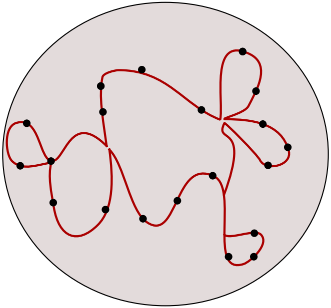


Fact

Given a tour T, one can find a non-self-crossing tour T' in polynomial time that has not larger cost.

Non-self-crossing

Definition: **Non-self-crossing** closed walk.
Definition: A tour is **non-self-crossing** if there is a non-self-crossing closed walk realizing it.



Fact

Given a tour T, one can find a non-self-crossing tour T' in polynomial time that has not larger cost.

Partial solutions

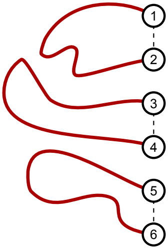
General idea: build larger and larger partial solutions.

Held-Karp algorithm: the partial solutions are $v_{\text{start}} - v$ paths visiting a subset T' of cities.

Partial solutions

General idea: build larger and larger partial solutions.

Held-Karp algorithm: the partial solutions are $v_{\text{start}} - v$ paths visiting a subset T' of cities.



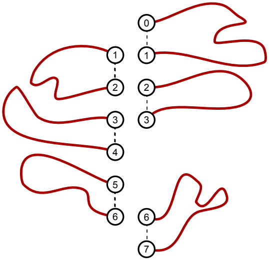
Generalization: a partial solution is a set of at most d pairwise disjoint paths with specified endpoints.

The **type** of a partial solution can be described by

- the set of endpoints of the paths,
- a matching between the endpoints, and
- the subset T' of visited cities.

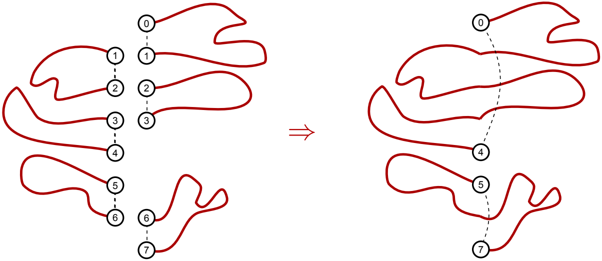
Merging partial solutions

Two compatible partial solutions can be merged in an obvious way:



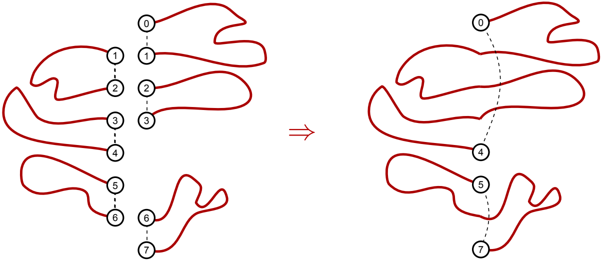
Merging partial solutions

Two compatible partial solutions can be merged in an obvious way:



Merging partial solutions

Two compatible partial solutions can be merged in an obvious way:



Algorithm

- Start with an initial set of trivial partial solutions.
- Combine two partial solutions as long as possible.
- Keep at most one partial solution from each type: the best one encountered so far.
- Return the best partial solution that consists of a single path (cycle) visiting all vertices.

Running time

Algorithm

- Start with an initial set of trivial partial solutions.
- Combine two partial solutions as long as possible.
- Keep at most one partial solution from each type: the best one encountered so far.
- Return the best partial solution that consists of a single path (cycle) visiting all vertices.

Running time

Algorithm

- Start with an initial set of trivial partial solutions.
- Combine two partial solutions as long as possible.
- Keep at most one partial solution from each type: the best one encountered so far.
- Return the best partial solution that consists of a single path (cycle) visiting all vertices.

With careful implementation, the running time is dominated by the number of types, whose number has two factors:

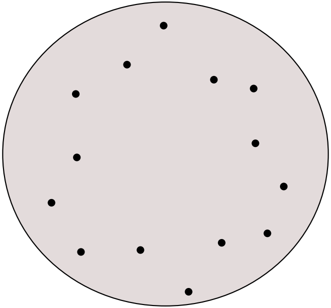
- endpoints described by at most d pairs of vertices
  $\Rightarrow k^{2d}$ possibilities,
- describing the subset T' of visited cities
  $\Rightarrow 2^k$ possibilities.

We can increase d up to $O(\sqrt{k})$, but we need to reduce somehow the number of possible subsets of cities!

Restricting the subset of cities

We restrict attention to a collection $\mathcal{T}$ of subsets of cities and consider only partial solutions that visit a subset in $\mathcal{T}$.

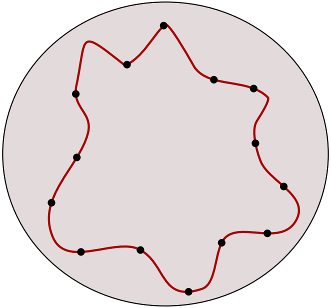
We need: a collection $\mathcal{T}$ of size $k^{O(\sqrt{k})}$ that guarantees finding an optimum solution.



Restricting the subset of cities

We restrict attention to a collection $\mathcal{T}$ of subsets of cities and consider only partial solutions that visit a subset in $\mathcal{T}$.

We need: a collection $\mathcal{T}$ of size $k^{O(\sqrt{k})}$ that guarantees finding an optimum solution.



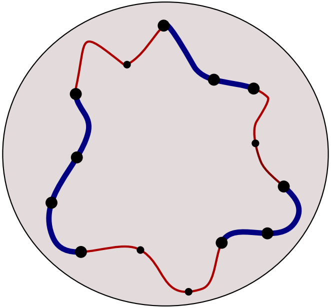
Definition of $\mathcal{T}$:

- Find a non-self-crossing 4-OPT tour.

Restricting the subset of cities

We restrict attention to a collection $\mathcal{T}$ of subsets of cities and consider only partial solutions that visit a subset in $\mathcal{T}$.

We need: a collection $\mathcal{T}$ of size $k^{O(\sqrt{k})}$ that guarantees finding an optimum solution.



Definition of $\mathcal{T}$:

- Find a non-self-crossing 4-OPT tour.
- A subset is in $\mathcal{T}$ if and only if it induces $O(\sqrt{k})$ consecutive intervals on the non-self-crossing 4-OPT tour.

Main result

Definition of $\mathcal{T}$:

- Find a non-self-crossing 4-OPT tour.
- A subset is in $\mathcal{T}$ if and only if it induces $O(\sqrt{k})$ consecutive intervals on the non-self-crossing 4-OPT tour.

Theorem

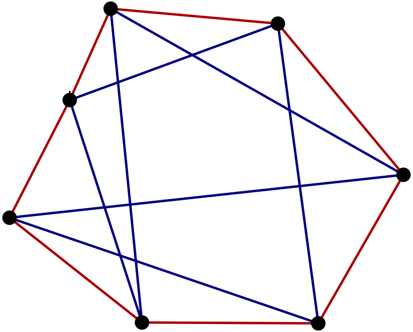
After setting $\mathcal{T}$ as above and $d = O(\sqrt{k})$, the Algorithm finds an optimum solution for Subset TSP on planar graphs.

Corollary

Subset TSP for k cities in a planar graph can be solved in time $(2^{O(\sqrt{k}\log k)} + W) \cdot n^{O(1)}$ if the weights are integers at most W.

The treewidth bound

Consider the union of an optimum solution and a 4-OPT solution as a graph on k vertices:



Lemma

For every non-self-crossing 4-OPT solution, there is an optimum solution such that their union has treewidth $O(\sqrt{k})$.

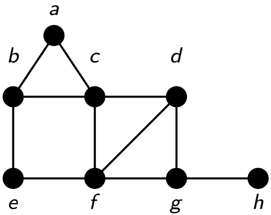
Treewidth

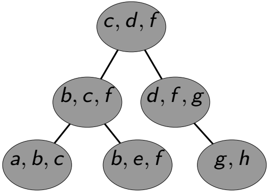
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every v, the bags containing v form a connected subtree.

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.

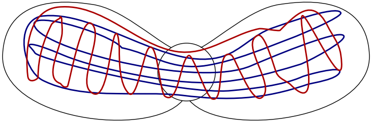




The treewidth bound

Lemma

For every non-self-crossing 4-OPT solution, there is an optimum solution such that their union has treewidth $O(\sqrt{k})$.

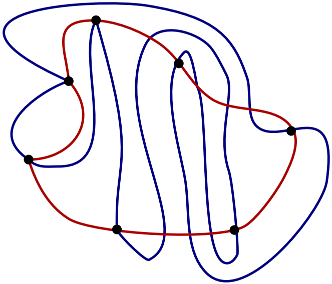


- The union has separators of size $O(\sqrt{k})$.
- In each component, the set of cities visited by the optimum solution is nice: it is the same as what $O(\sqrt{k})$ segments of the 4-OPT tour visited.
- We can use this tree decomposition to prove that the Algorithm finds an optimum solution.

Proof of the treewidth bound

Consider the closed walk corresponding to the 4-OPT solution and pick an optimum solution and a closed walk representing that.

The **union** is a planar graph:

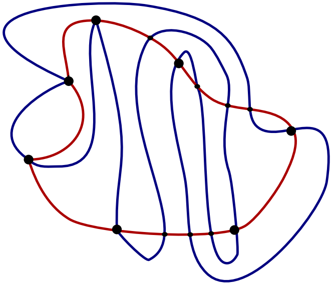


Select the optimum solution and the closed walk such that the two tours cross each other the minimum number of times.

Proof of the treewidth bound

Consider the closed walk corresponding to the 4-OPT solution and pick an optimum solution and a closed walk representing that.

The **union** is a planar graph:



We give an $O(\sqrt{k})$ bound on the treewidth of this planar graph

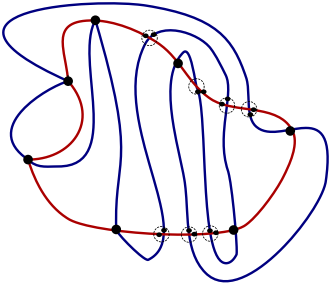
$\Downarrow$

A $O(\sqrt{k})$ bound follows for the k-vertex graph, as it is a minor of this graph after duplicating the vertices.

Proof of the treewidth bound

Consider the closed walk corresponding to the 4-OPT solution and pick an optimum solution and a closed walk representing that.

The **union** is a planar graph:



We give an $O(\sqrt{k})$ bound on the treewidth of this planar graph

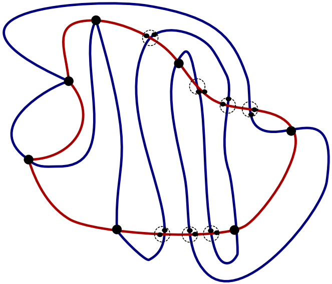
$\Downarrow$

A $O(\sqrt{k})$ bound follows for the k-vertex graph, as it is a minor of this graph after duplicating the vertices.

Proof of the treewidth bound

Consider the closed walk corresponding to the 4-OPT solution and pick an optimum solution and a closed walk representing that.

The **union** is a planar graph:



We prove that every 3-connected component of the planar graph has $O(k)$ vertices

$\Downarrow$

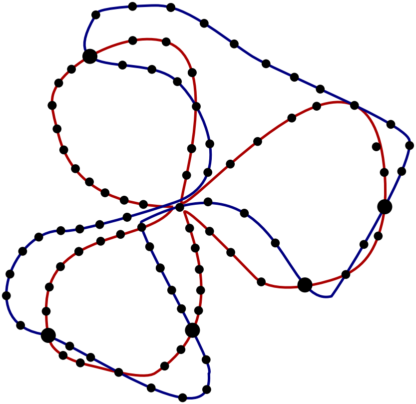
$O(\sqrt{k})$ treewidth bound on the 3-connected components

$\Downarrow$

same bound for the whole graph.

Representations

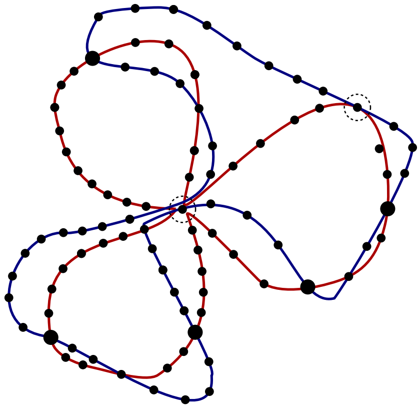
The union of of the 4-OPT solution and the optimum solution can be degenerate in several ways (two tours share edges, touch each other, revisit vertices etc.).



We work with a **representation** of the union, which is a 4-regular planar graph where every vertex (except the cities) is a crossing.

Representations

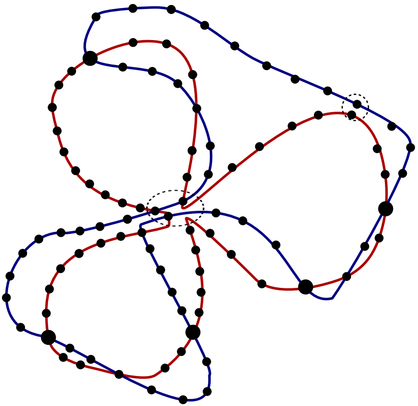
The union of of the 4-OPT solution and the optimum solution can be degenerate in several ways (two tours share edges, touch each other, revisit vertices etc.).



We work with a **representation** of the union, which is a 4-regular planar graph where every vertex (except the cities) is a crossing.

Representations

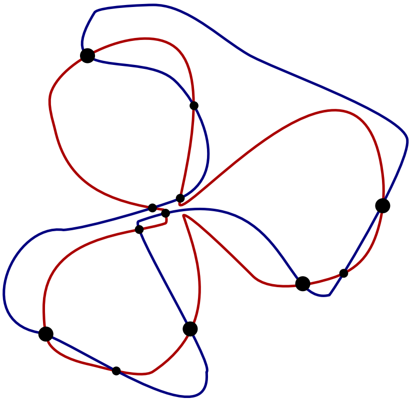
The union of of the 4-OPT solution and the optimum solution can be degenerate in several ways (two tours share edges, touch each other, revisit vertices etc.).



We work with a **representation** of the union, which is a 4-regular planar graph where every vertex (except the cities) is a crossing.

Representations

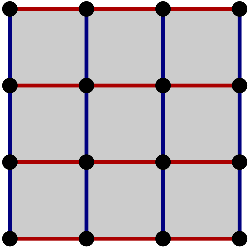
The union of of the 4-OPT solution and the optimum solution can be degenerate in several ways (two tours share edges, touch each other, revisit vertices etc.).



We work with a **representation** of the union, which is a 4-regular planar graph where every vertex (except the cities) is a crossing.

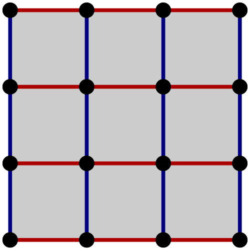
Grids

A **grid** is a 16-vertex subgraph of the representation of the union of the 4-OPT solution and the optimum solution:



Grids

A **grid** is a 16-vertex subgraph of the representation of the union of the 4-OPT solution and the optimum solution:



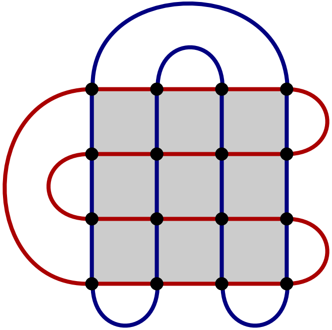
Lemma

If a 3-connected component of the representation has size $\Omega(k)$, then there is a grid.

Proof idea: 4-regular and $O(k)$ faces have length < 4
$\Rightarrow$ Euler's formula implies that most of the faces have length 4
$\Rightarrow$ a 4-face surrounded by 4-faces should be a grid.

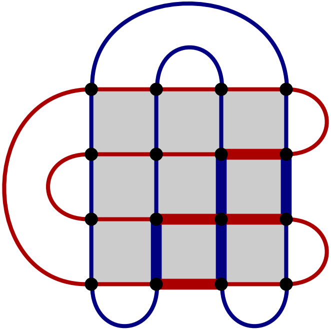
Grids

Suppose that the grid is used like this by two tours:



Grids

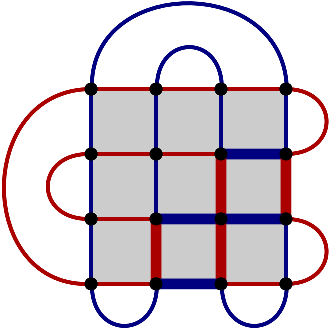
Suppose that the grid is used like this by two tours:



- Let us exchange these two sets of edges between the two tours.

Grids

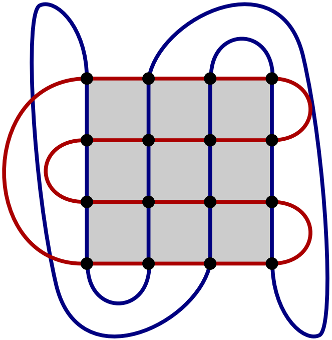
Suppose that the grid is used like this by two tours:



- Let us exchange these two sets of edges between the two tours.
- The 4-OPT tour cannot improve.
- The optimum tour cannot improve.
- We get another optimum tour such that the representation has fewer crossings.

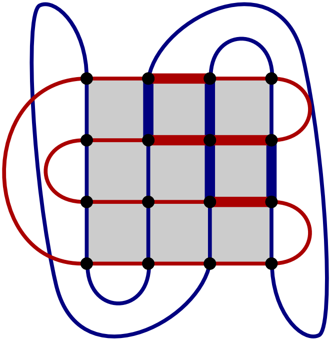
Grids — other cases:

C type + S type:



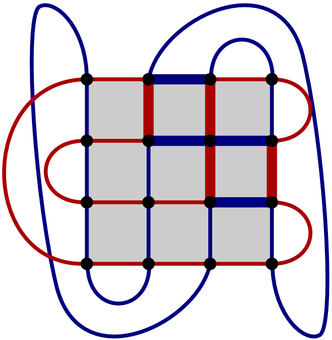
Grids — other cases:

C type + S type:



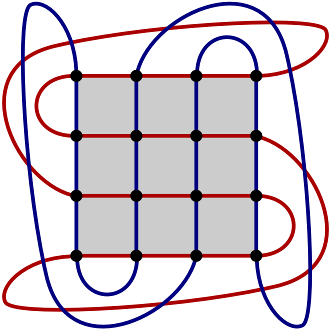
Grids — other cases:

C type + S type:



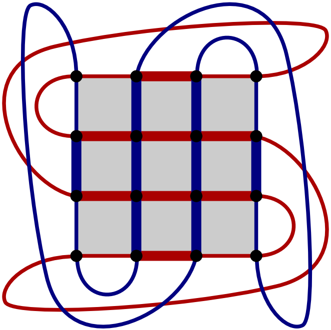
Grids — other cases:

S type + S type:



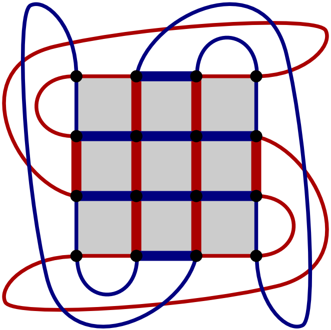
Grids — other cases:

S type + S type:



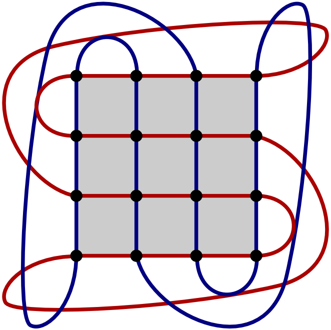
Grids — other cases:

S type + S type:



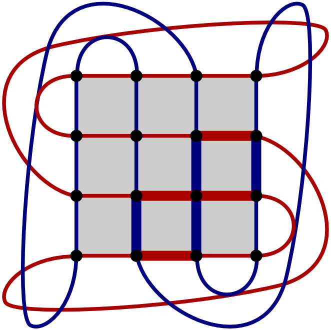
Grids — other cases:

S type + inverted S type:



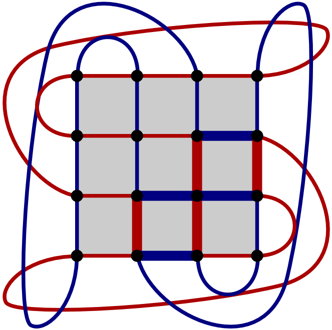
Grids — other cases:

S type + inverted S type:



Grids — other cases:

S type + inverted S type:



Overview

- Algorithm:
  - Find a 4-OPT tour.
  - Partial solutions: $O(\sqrt{k})$ disjoint paths, visiting $O(\sqrt{k})$ consecutive intervals on the 4-OPT tour.
  - Merge partial solutions until the optimum solution is found.
- Treewidth bound: the union of the 4-OPT tour and some optimum tour is a k-vertex graph with treewidth $O(\sqrt{k})$.
  - Study the union in the planar graph.
  - Every 3-connected component has $O(k)$ vertices, otherwise there is a grid and an exchange argument could be used.
  - Union in the planar graph has treewidth $O(\sqrt{k}) \Rightarrow$ the k-vertex graph has treewidth $O(\sqrt{k})$.