On subexponential parameterized algorithms for Steiner Tree and Directed Subset TSP on planar graphs

Dániel Marx[1] Marcin Pilipczuk[2] Michał Pilipczuk[2]

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

[2]Institute of Informatics,
University of Warsaw, Poland

FOCS 2018
Paris, France
October 9, 2018

Square root phenomenon

NP-hard problems become easier on planar graphs, and usually exactly by a square root factor.

Square root phenomenon

NP-hard problems become easier on planar graphs, and usually exactly by a square root factor.

The running time is still exponential, but significantly smaller:

$$
\begin{aligned}
2^{O(n)} &\Rightarrow 2^{O(\sqrt{n})} \\
n^{O(k)} &\Rightarrow n^{O(\sqrt{k})} \\
2^{O(k)} \cdot n^{O(1)} &\Rightarrow 2^{O(\sqrt{k})} \cdot n^{O(1)}
\end{aligned}
$$

Square root phenomenon

NP-hard problems become easier on planar graphs, and usually exactly by a square root factor.

The running time is still exponential, but significantly smaller:

$$2^{O(n)} \Rightarrow 2^{O(\sqrt{n})}$$
$$n^{O(k)} \Rightarrow n^{O(\sqrt{k})}$$
$$2^{O(k)} \cdot n^{O(1)} \Rightarrow 2^{O(\sqrt{k})} \cdot n^{O(1)}$$

Several known examples known where such improvement is possible, and (assuming the ETH)

- $O(k)$ is best possible for general graphs and
- $O(\sqrt{k})$ is best possible for planar graphs.

Two standard techniques

1. **Using treewidth:**
   Works for e.g. 3-COLORING or HAMILTONIAN CYCLE:

Planar graphs have treewidth $O(\sqrt{n})$	$+$	$2^{O(w)} \cdot n^{O(1)}$ algorithm for treewidth w	$\Rightarrow$	$2^{O(\sqrt{n})}$ algorithm

Two standard techniques

1. **Using treewidth:**
   Works for e.g. 3-Coloring or Hamiltonian Cycle:

Planar graphs have treewidth $O(\sqrt{n})$	+	$2^{O(w)} \cdot n^{O(1)}$ algorithm for treewidth w	$\Rightarrow$	$2^{O(\sqrt{n})}$ algorithm

2. **Bidimensionality:**
   Works for e.g. k-Path or Vertex Cover:

Trivial answer if treewidth is $\Omega(\sqrt{k})$.	+	$2^{O(w)} \cdot n^{O(1)}$ algorithm for treewidth w	$\Rightarrow$	$2^{O(\sqrt{k})} \cdot n^{O(1)}$ algorithm

Other results

Many other result were obtained using problem-specific techniques:

- Strongly Connected Steiner Subgraph [Chitnis et al. 2014]
- Multiway Cut [Klein and M. 2012], [Colin de Verdière 2017]
- Subgraph Isomorphism for connected bounded-degree patterns [Fomin et al. 2016]
- Subset TSP [Klein and M. 2014]
- Facility Location [M. and Pilipczuk 2015]
- Odd Cycle Transversal [Lokshtanov et al. 2012]

Two main results

1. A positive result:

Directed Subset TSP with k terminals can be solved

- in time $2^{O(k)} \cdot n^{O(1)}$ in **general** graphs, [Held-Karp 1962]
- in time $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$ in **planar** graphs. **[new result #1]**

Two main results

1. A positive result:

DIRECTED SUBSET TSP with k terminals can be solved

- in time $2^{O(k)} \cdot n^{O(1)}$ in **general** graphs,
  [Held-Karp 1962]
- in time $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$ in **planar** graphs.
  [new result #1]

2. A negative result:

STEINER TREE with k terminals

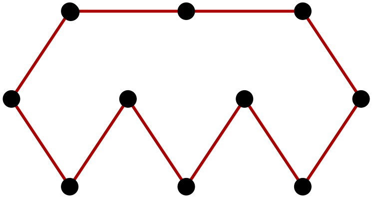
- can be solved in time $2^{O(k)} \cdot n^{O(1)}$ in **general** graphs,
  [Dreyfus and Wagner 1971]
- cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$ in **planar undirected** graphs (assuming the ETH). **[new result #2]**

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]

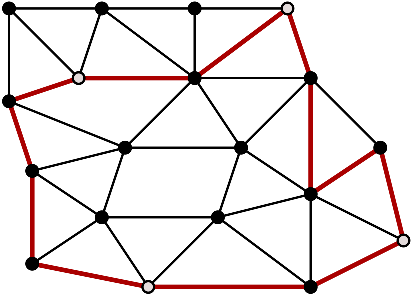
TSP with n cities can be solved in time $O(2^n \cdot n^2)$.

Dynamic programming:

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

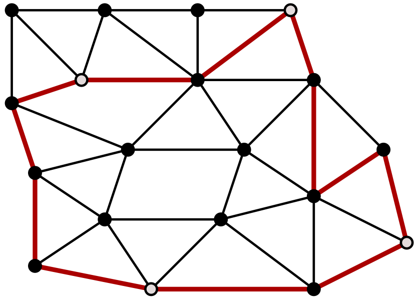
SUBSET TSP on planar graphs

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.

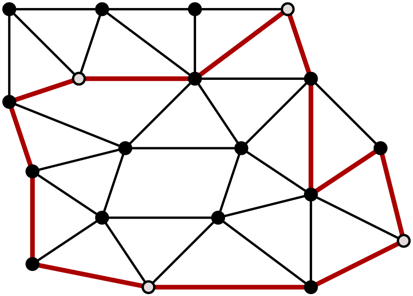


- Can be solved in time $n^{O(\sqrt{n})}$.
- Can be solved in time $2^k \cdot n^{O(1)}$.

Question: Can we restrict the exponential dependence to k **and** exploit planarity?

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.

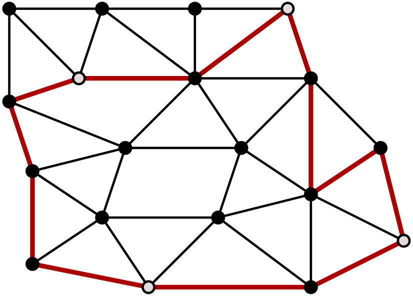


Theorem [Klein and M. 2014]

Subset TSP for k cities in a unit-weight undirected 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 [new result #1]

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

Partial solutions

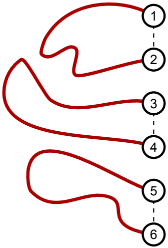
General idea: build larger and larger partial solutions.

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.



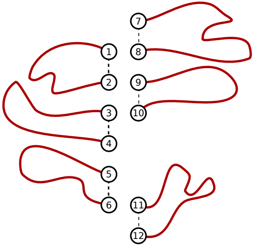
Generalization: a partial solution is a set of at most d pairwise disjoint paths with specified cities as endpoints.

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.

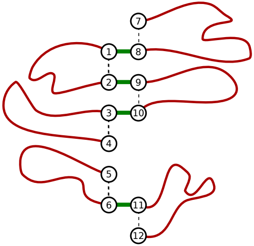
Merging partial solutions

Two partial solutions can be merged in an obvious way if a matching is given between the endpoints:



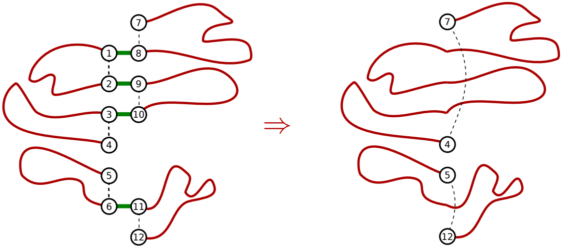
Merging partial solutions

Two partial solutions can be merged in an obvious way if a matching is given between the endpoints:



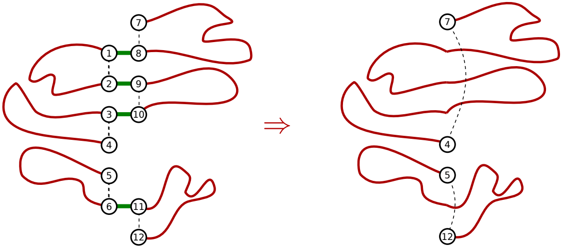
Merging partial solutions

Two partial solutions can be merged in an obvious way if a matching is given between the endpoints:



Merging partial solutions

Two partial solutions can be merged in an obvious way if a matching is given between the endpoints:



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.

For $d = O(\sqrt{k})$, the number of types ($\approx$ running time) is

endpoints of $O(\sqrt{k})$ paths

subset $T' \subseteq T$ of visited cities

$$k^{O(\sqrt{k})} \cdot 2^k$$

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.

For $d = O(\sqrt{k})$, the number of types ($\approx$ running time) is

endpoints of $O(\sqrt{k})$ paths　　　　subset $T' \subseteq T$ of visited cities

$$k^{O(\sqrt{k})} \cdot 2^k$$

We need to reduce somehow the number of possible subsets of cities partial solutions can visit!

Running time

Algorithm

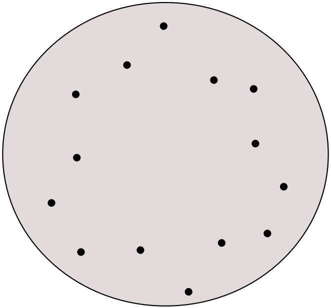
- Start with an initial set of trivial partial solutions.
- Combine two partial solutions as long as possible.
- Keep at most one partial solution from each type: the best one encountered so far.
- Return the best partial solution that consists of a single path (cycle) visiting all vertices.

Basic idea

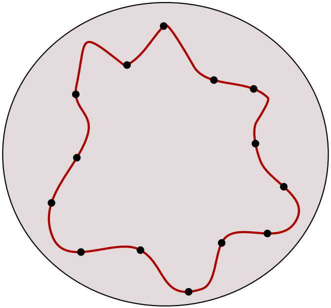
We restrict attention to a collection $\mathcal{T}$ of subsets of cities and consider only partial solutions that visit a subset in $\mathcal{T}$.

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

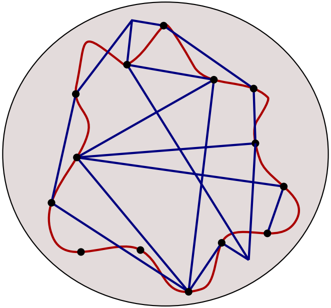
Bounding the treewidth . . . of what?



Bounding the treewidth . . . of what?



Bounding the treewidth ...of what?

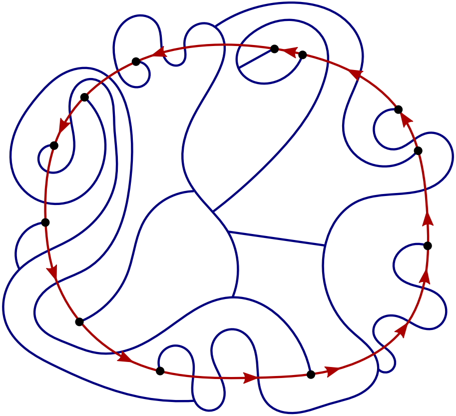


The following principle can be deduced from earlier work:

Exploit that the union of the unknown solution + a known something has treewidth $O(\sqrt{k})$.

Bounding treewidth

Take an arbitrary Steiner tree T and assume first that it intersects OPT $O(k)$ times.

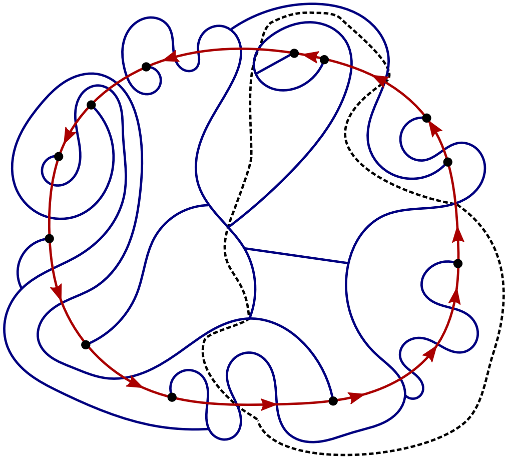


$OPT + T$ has $O(k)$ branch vertices
$\Rightarrow$ treewidth $O(\sqrt{k})$
$\Rightarrow$ exists a sphere-cut decomposition of width $O(\sqrt{k})$

Sphere-cut decompositions

Noose: a closed curve intersecting the graph only at vertices.

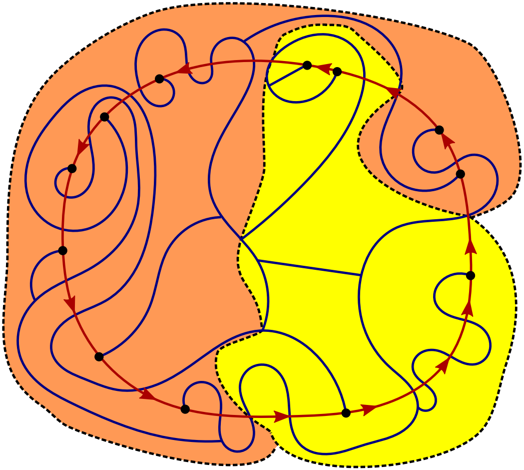
Sphere-cut decomposition of width $O(\sqrt{k})$: a recursive decomposition where the boundary of each part is a noose intersecting $O(\sqrt{k})$ vertices.



Sphere-cut decompositions

Noose: a closed curve intersecting the graph only at vertices.

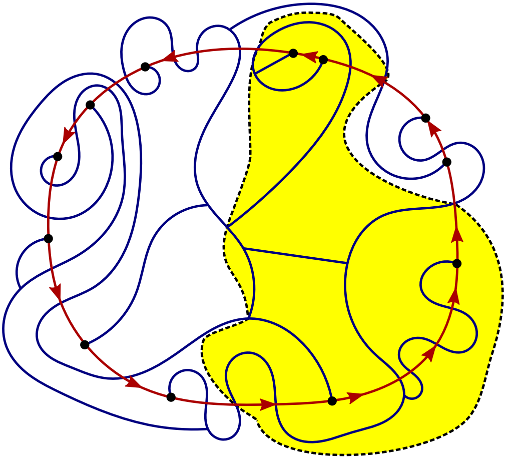
Sphere-cut decomposition of width $O(\sqrt{k})$: a recursive decomposition where the boundary of each part is a noose intersecting $O(\sqrt{k})$ vertices.



Sphere-cut decompositions

Noose: a closed curve intersecting the graph only at vertices.

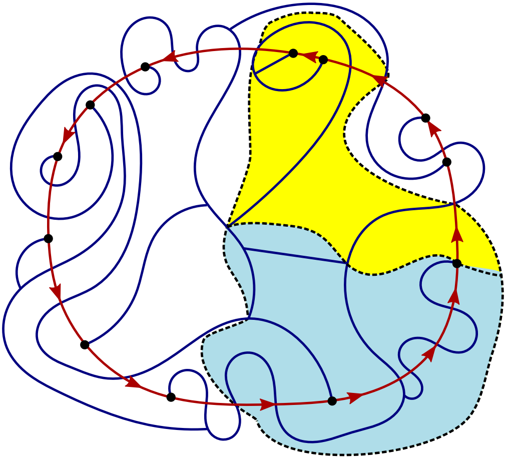
Sphere-cut decomposition of width $O(\sqrt{k})$: a recursive decomposition where the boundary of each part is a noose intersecting $O(\sqrt{k})$ vertices.



Sphere-cut decompositions

Noose: a closed curve intersecting the graph only at vertices.

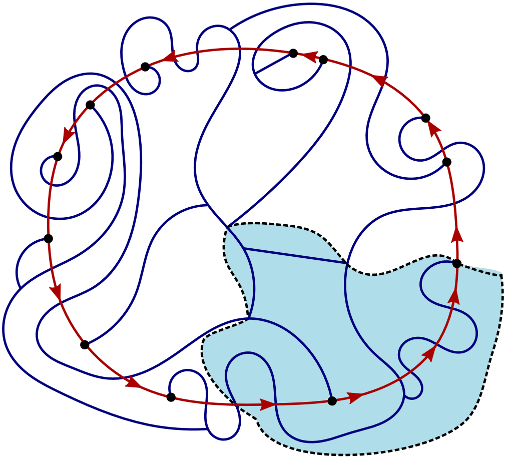
Sphere-cut decomposition of width $O(\sqrt{k})$: a recursive decomposition where the boundary of each part is a noose intersecting $O(\sqrt{k})$ vertices.



Sphere-cut decompositions

Noose: a closed curve intersecting the graph only at vertices.

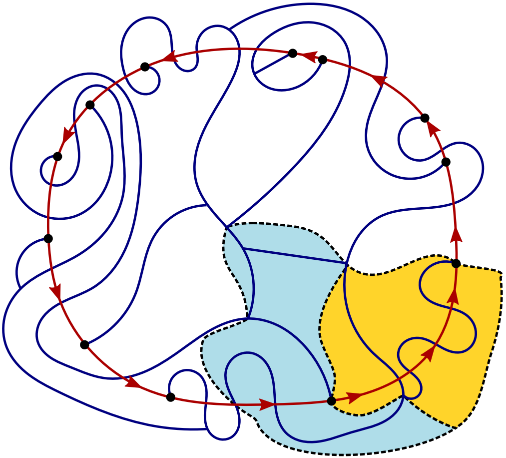
Sphere-cut decomposition of width $O(\sqrt{k})$: a recursive decomposition where the boundary of each part is a noose intersecting $O(\sqrt{k})$ vertices.



Sphere-cut decompositions

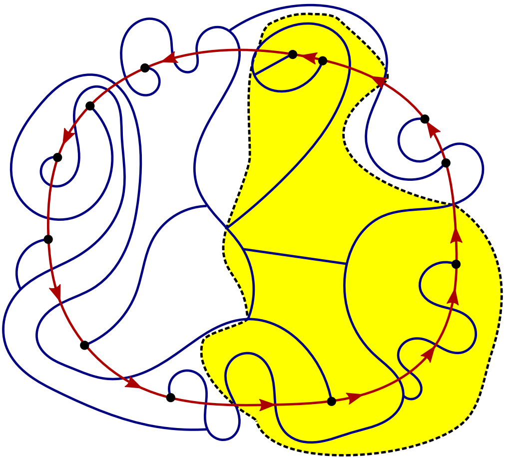
Noose: a closed curve intersecting the graph only at vertices.

Sphere-cut decomposition of width $O(\sqrt{k})$**:** a recursive decomposition where the boundary of each part is a noose intersecting $O(\sqrt{k})$ vertices.



Partial solutions

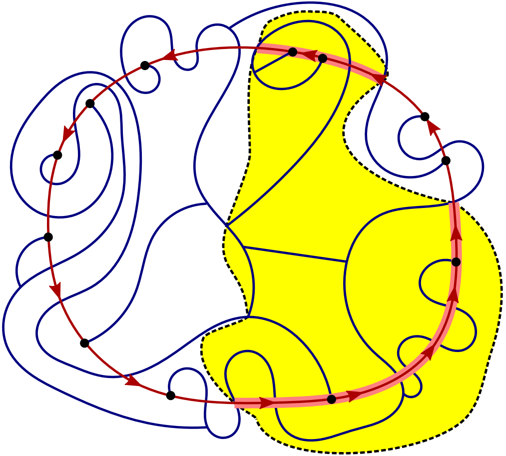
Each noose cuts out a partial solution with $O(\sqrt{k})$ subpaths of OPT.



What can be the set of terminals visited by this partial solution?

Partial solutions

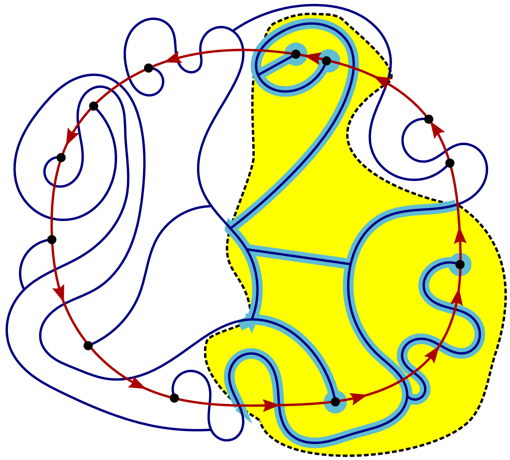
Each noose cuts out a partial solution with $O(\sqrt{k})$ subpaths of OPT.



What can be the set of terminals visited by this partial solution?

Partial solutions

Each noose cuts out a partial solution with $O(\sqrt{k})$ subpaths of OPT.

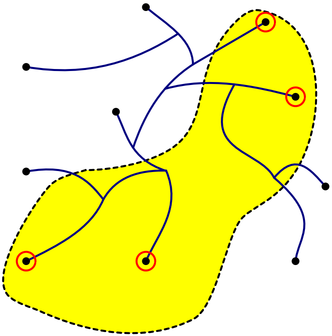


What can be the set of terminals visited by this partial solution?

Cutting terminals from a tree

Lemma

We can compute a collection $\mathcal{T}$ of $k^{O(\sqrt{k})}$ subsets of terminals such that if C is a cycle intersecting the tree T at most $O(\sqrt{k})$ times, then the set of terminals enclosed by C is in $\mathcal{T}$.



We can restrict attention only to partial solutions restricted to $\mathcal{T}$!

Algorithm

Algorithm

- Compute the collection $\mathcal{T}$ (possible sets of terminals enclosed by a cycle intersecting tree T at most $O(\sqrt{k})$ times).
- Start with an initial set of trivial partial solutions.
- Combine two partial solutions as long as possible and keep it only if it visits a subset in $\mathcal{T}$.
- 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.

Only $k^{O(k)}$ subproblems are considered

$$\Downarrow$$

Running time is $k^{O(k)} n^{O(1)}$.

Algorithm

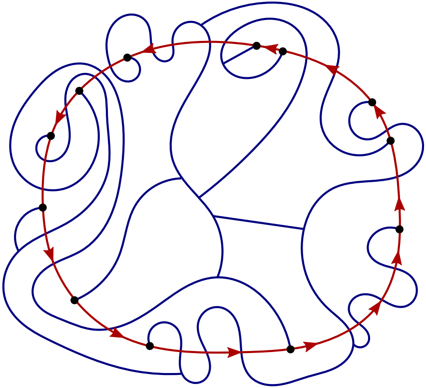
Algorithm

- Compute the collection $\mathcal{T}$ (possible sets of terminals enclosed by a cycle intersecting tree T at most $O(\sqrt{k})$ times).
- Start with an initial set of trivial partial solutions.
- Combine two partial solutions as long as possible and keep it only if it visits a subset in $\mathcal{T}$.
- 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.

Existence of the sphere-cut decomposition implies that the algorithm finds an optimum solution!

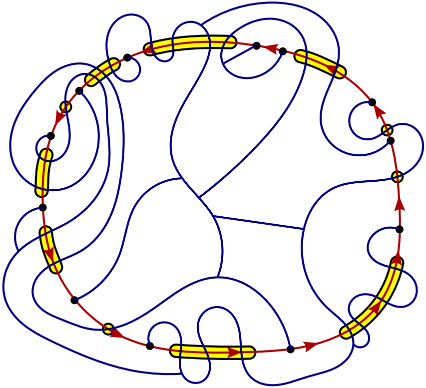
Many intersections

What happens if $OPT + T$ has more than $O(k)$ intersections?



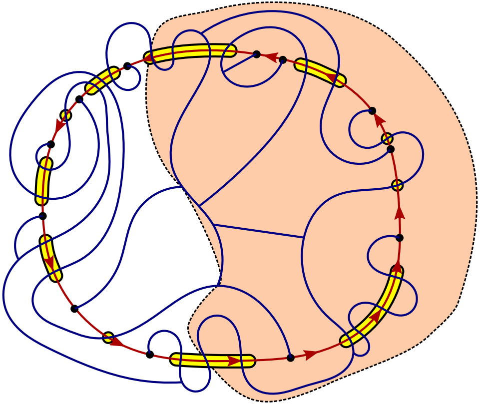
Many intersections

What happens if $OPT + T$ has more than $O(k)$ intersections?



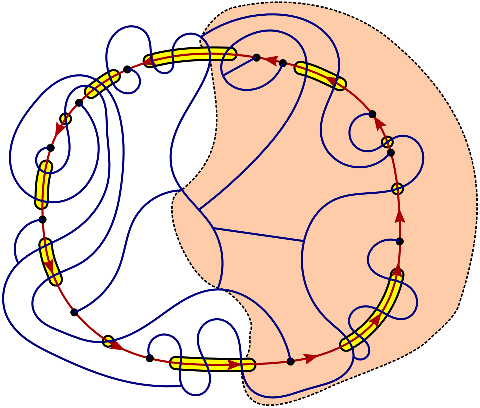
Many intersections

What happens if $OPT + T$ has more than $O(k)$ intersections?



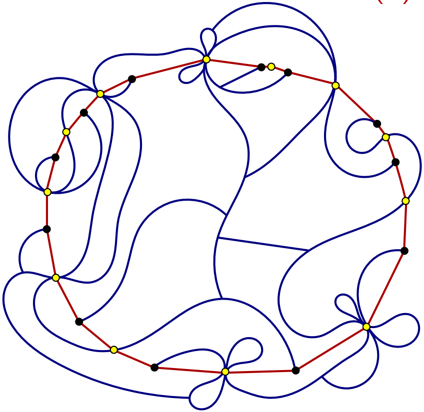
Many intersections

What happens if $OPT + T$ has more than $O(k)$ intersections?



Many intersections

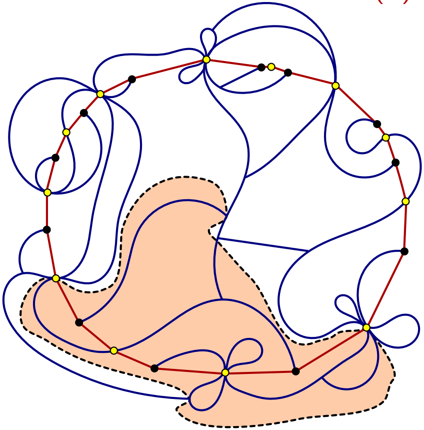
What happens if $OPT + T$ has more than $O(k)$ intersections?



- Let us contract the subpaths of OPT between consecutive terminals (each such path is a shortest path).
- Each noose goes through $O(\sqrt{k})$ contracted vertices
  $\Rightarrow$ we can guess the contractions that produced these vertices.

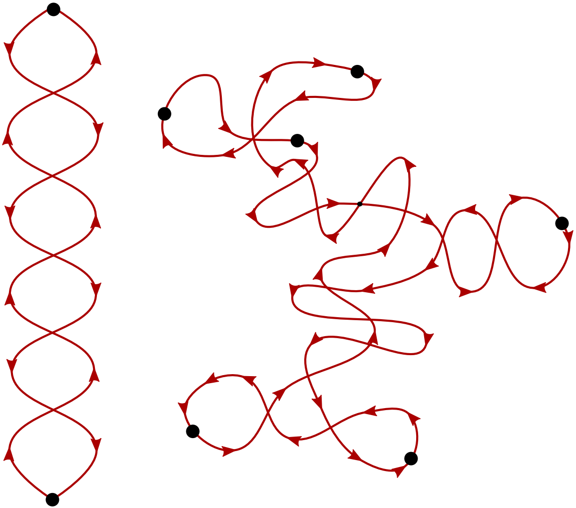
Many intersections

What happens if $OPT + T$ has more than $O(k)$ intersections?



- Let us contract the subpaths of OPT between consecutive terminals (each such path is a shortest path).
- Each noose goes through $O(\sqrt{k})$ contracted vertices
  $\Rightarrow$ we can guess the contractions that produced these vertices.

Self-crossing solutions



It is not possible to bound the number of self-crossings by a function of k, but we can show that there is a solution that is a "cactus."

Lower bound for Steiner Tree

Theorem [new result #2]

Assuming the ETH, Steiner Tree on **planar undirected** graphs with k terminals cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$.

Standard techniques show that Steiner Tree (and many other problems) do not have $2^{o(\sqrt{k})} \cdot n^{O(1)}$ time algorithms assuming the ETH, but a lower bound ruling out $2^{o(k)} \cdot n^{O(1)}$ is quite unusual!

Standard lower bounds for planar problems

ETH + Sparsification Lemma

There is no $2^{o(n+m)}$-time algorithm for m-clause 3SAT.

- Typical reduction from 3SAT creates $O(n+m)$ gadgets and $O((n+m)^2)$ crossings in the plane.
- A crossing typically increases the size by $O(1)$.

3SAT formula ϕ n variables m clauses	$\Rightarrow$	Planar graph G' $O((n+m)^2)$ vertices $O((n+m)^2)$ edges

Corollary

Assuming the ETH, there is no $2^{o(\sqrt{n})}$ algorithm for Steiner Tree on an n-vertex planar graph.

Standard lower bounds for planar problems

ETH + Sparsification Lemma

There is no $2^{o(n+m)}$-time algorithm for m-clause 3SAT.

- Typical reduction from 3SAT creates $O(n+m)$ gadgets and $O((n+m)^2)$ crossings in the plane.
- A crossing typically increases the size by $O(1)$.

3SAT formula ϕ		Planar graph G'
n variables	$\Rightarrow$	$O((n+m)^2)$ vertices
m clauses		$O((n+m)^2)$ edges

Corollary

Assuming the ETH, there is no $2^{o(\sqrt{k})} \cdot n^{O(1)}$ algorithm for STEINER TREE on an n-vertex planar graph with k terminals.

Standard lower bounds for planar problems

ETH + Sparsification Lemma

There is no $2^{o(n+m)}$-time algorithm for m-clause $\mathrm{3SAT}$.

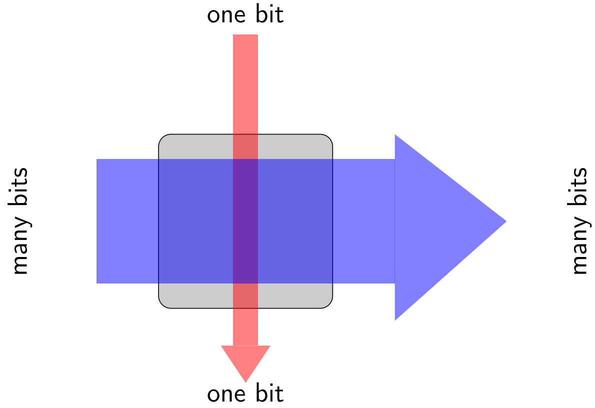
- Typical reduction from $\mathrm{3SAT}$ creates $O(n+m)$ gadgets and $O((n+m)^2)$ crossings in the plane.
- A crossing typically increases the size by $O(1)$.

$\mathrm{3SAT}$ formula ϕ		Planar graph G'
n variables	$\Rightarrow$	$O((n+m)^2)$ vertices
m clauses		$O((n+m)^2)$ edges

No way such reductions could give a bound stronger than $2^{o(\sqrt{k})}$!

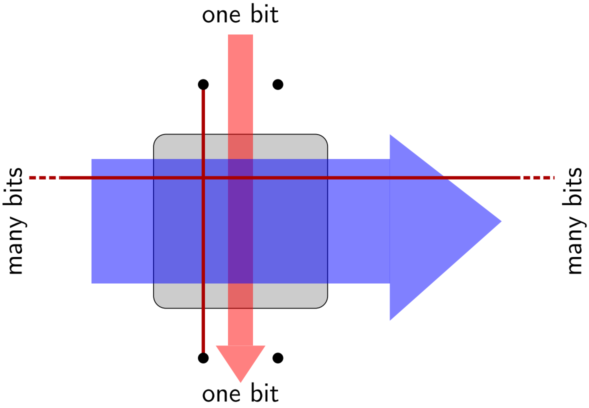
Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.



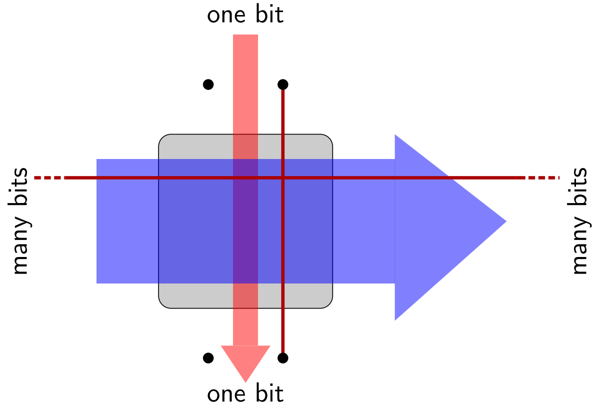
Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.



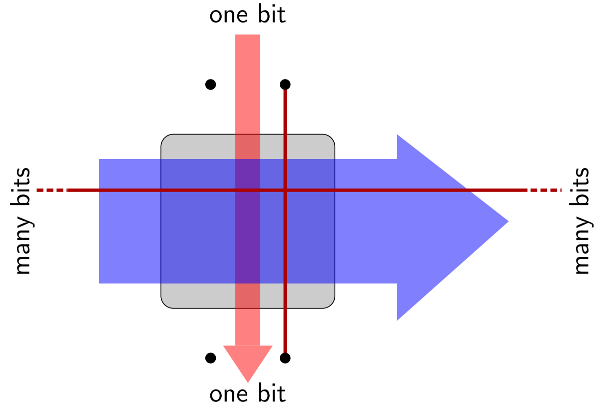
Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.



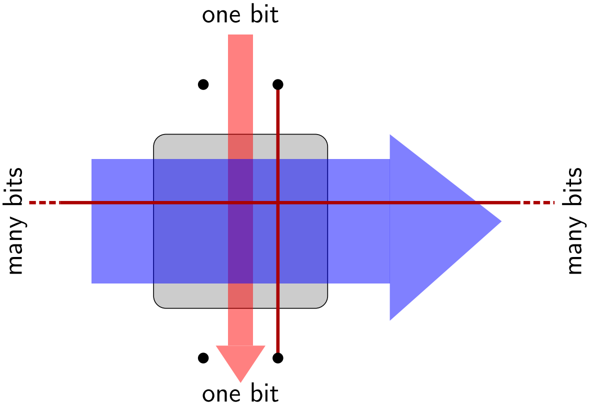
Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.



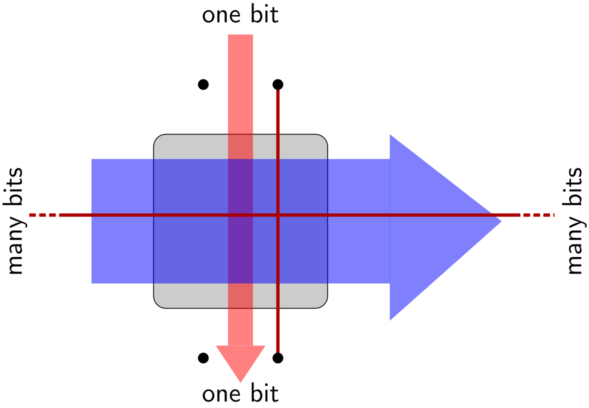
Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.



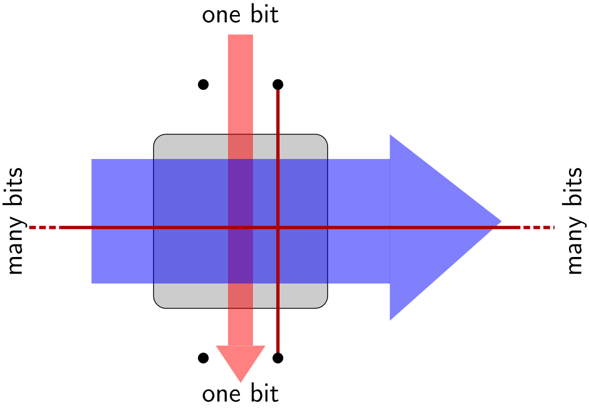
Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.



Stronger lower bound

We get around this issue by crossing gadgets where a stream of **many bits** cross a stream of **one bit** and has only $O(1)$ terminals.

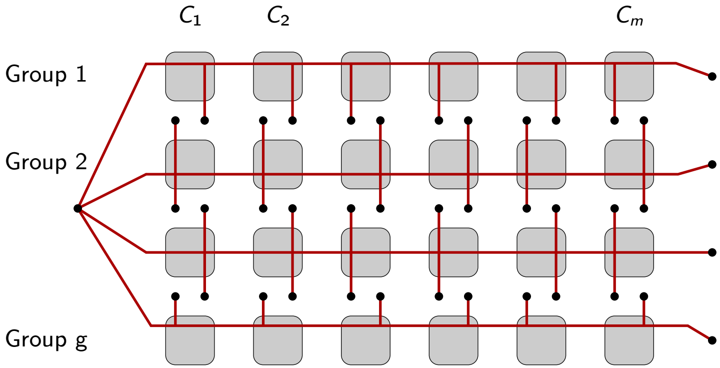


Reduction from 3SAT

Partition the variables into g groups of size n/g each.

- **Horizontal flow:** assignment in group i ($2^{n/g}$ possibilities)
- **Vertical flow:** checking satisfiability of each clause C_j.

Graph size: $N = 2^{O(n/g)}$ with $k = O(m \cdot g)$ terminals.

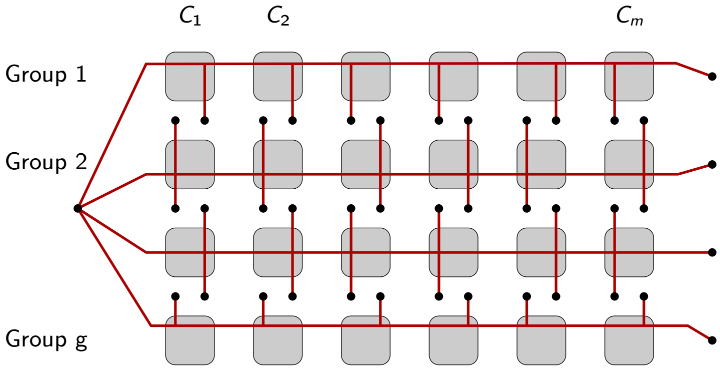


Reduction from 3SAT

Partition the variables into g groups of size n/g each.

- **Horizontal flow:** assignment in group i ($2^{n/g}$ possibilities)
- **Vertical flow:** checking satisfiability of each clause C_j.

Graph size: $N = 2^{O(n/g)}$ with $k = O(m \cdot g)$ terminals.

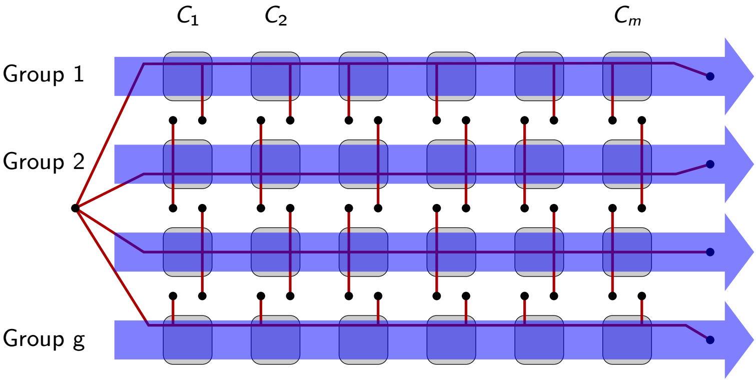


Reduction from 3SAT

Partition the variables into g groups of size n/g each.

- **Horizontal flow:** assignment in group i ($2^{n/g}$ possibilities)
- **Vertical flow:** checking satisfiability of each clause C_j.

Graph size: $N = 2^{O(n/g)}$ with $k = O(m \cdot g)$ terminals.

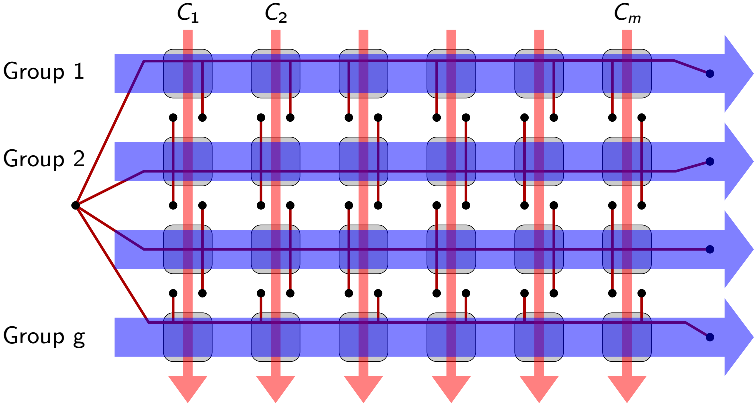


Reduction from 3SAT

Partition the variables into g groups of size n/g each.

- **Horizontal flow:** assignment in group i ($2^{n/g}$ possibilities)
- **Vertical flow:** checking satisfiability of each clause C_j.

Graph size: $N = 2^{O(n/g)}$ with $k = O(m \cdot g)$ terminals.



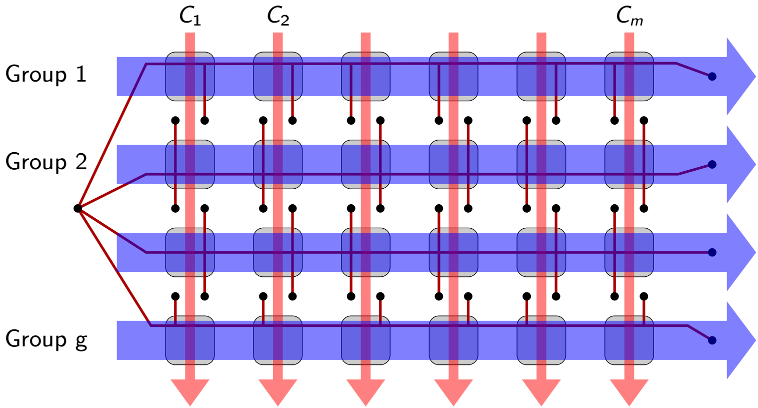
Reduction from 3SAT

Graph size: $N = 2^{O(n/g)}$ with $k = O(m \cdot g)$ terminals.

Running time $2^{O(k/g^2)} \cdot N^{O(1)}$ for Steiner Tree

$\Downarrow$

Running time $2^{O(m/g)} \cdot 2^{O(n/g)} = 2^{o(n+m)}$ for 3SAT



Summary

1. **Main positive result**

   SUBSET TSP for k cities in a **directed** planar graph can be solved in time $2^{O(\sqrt{k}\log k)} \cdot n^{O(1)}$.

Exploit that the union of the unknown solution + a known something has treewidth $O(\sqrt{k})$.

2. **Main negative result**

   Assuming the ETH, STEINER TREE on **planar undirected** graphs with k terminals cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$.

The square root phenomenon does not appear for every problem, making the previous positive results even more interesting!