Tight bounds for planar strongly connected Steiner subgraph with fixed number of terminals (and extensions)

Rajesh Chitnis[1] MohammadTaghi Hajiaghayi[1]
Dániel Marx[2]

[1]Computer Science Department
University of Maryland

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

SODA 2014
January 7, 2014
Portland, OR

Connecting terminals

Undirected graphs:

Steiner Tree	
Input:	An undirected graph G with terminals $t_1, \ldots, t_k$.
Find:	A tree T of G containing every t_i.
Goal:	Minimize the size of F.

A classical dynamic programming algorithm:

Theorem [Dreyfus and Wagner 1972]

Steiner Tree can be solved in time $3^k \cdot n^{O(1)}$.

Recent improvement:

Theorem [Björklund et al. 2007]

Steiner Tree can be solved in time $2^k \cdot n^{O(1)}$.

Connecting terminals

Directed graphs:

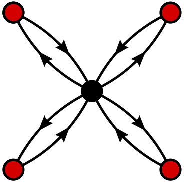
STRONGLY CONNECTED STEINER SUBGRAPH

Input: A directed graph G with terminals $t_1, \ldots, t_k$.
Find: A subgraph F of G such that there is a $t_i \to t_j$ path in F for every $1 \leq i, j \leq k$.
Goal: Minimize the size of F.

What is the complexity of STRONGLY CONNECTED STEINER SUBGRAPH for fixed k?

Edge vs. vertex versions

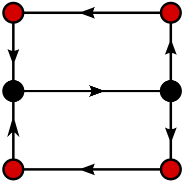
We can minimize either the number of edges or vertices — can lead to different optimum solutions.



5 vertices

8 edges

vs.



6 vertices

7 edges

We focus here on the vertex version (which is typically harder).

Strongly Connected Steiner Subgraph

Theorem

Strongly Connected Steiner Subgraph on general directed graphs

- can be solved in time $n^{O(k)}$ [Feldman and Ruhl 2006],
- is W[1]-hard parameterized by k [Guo, Niedermeier, Suchý 2011], thus an $f(k) \cdot n^{O(1)}$ algorithm is unlikely.

Strongly Connected Steiner Subgraph

Theorem

Strongly Connected Steiner Subgraph on general directed graphs

- can be solved in time $n^{O(k)}$ [Feldman and Ruhl 2006],
- is W[1]-hard parameterized by k [Guo, Niedermeier, Suchý 2011], thus an $f(k) \cdot n^{O(1)}$ algorithm is unlikely.

Revisiting the W[1]-hardness proof of [Guo, Niedermeier, Suchý 2011] more carefully gives:

Theorem

There is no $f(k) \cdot n^{o(k/\log k)}$ time algorithm for Strongly Connected Steiner Subgraph, unless the Exponential Time Hypothesis (ETH) fails.

[ETH: n-variable 3SAT cannot be solved in time $2^{o(n)}$.]

Planar graphs

- Parameterized problems are typically much easier on planar graphs.
- Bidimensionality theory or other techniques often give $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time algorithms.
- Do we get such an improvement for Strongly Connected Steiner Subgraph?

Planar graphs

- Parameterized problems are typically much easier on planar graphs.
- Bidimensionality theory or other techniques often give $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time algorithms.
- Do we get such an improvement for Strongly Connected Steiner Subgraph?

Main Result

Strongly Connected Steiner Subgraph on planar directed graphs

- can be solved in time $2^{O(k \log k)} \cdot n^{O(\sqrt{k})}$,
- has no $f(k) \cdot n^{o(\sqrt{k})}$ time algorithm (assuming ETH).

Upper bound:
The algorithm

Algorithm of Feldman and Ruhl

The Feldman-Ruhl game

- Let an arbitrary terminal be the root r.
- Put a forward pebble and a backward pebble on each of the remaining $k-1$ terminals ($2(k-1)$ pebbles in total).
- A set of legal moves and their cost are defined.

Algorithm of Feldman and Ruhl

The Feldman-Ruhl game

- Let an arbitrary terminal be the root r.
- Put a forward pebble and a backward pebble on each of the remaining $k-1$ terminals ($2(k-1)$ pebbles in total).
- A set of legal moves and their cost are defined.

The following equivalence is proved:

Theorem [Feldman and Ruhl 2006]

There is a sequence of legal moves with total cost C moving all the pebbles to the root r.

$$\Updownarrow$$

There is a solution of STRONGLY CONNECTED STEINER SUBGRAPH with C vertices.

The existence of the required sequence of moves can be tested in time $n^{O(k)}$.

Legal moves

- **Forward move:** a forward pebble at u moves on an edge $u \to v$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.
- **Backward move:** a backward pebble at u moves on an edge $v \to u$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.
- **Flip move:** Let f be a forward pebble at u, let b be a backward pebble at v, and let W be a $u \to v$ walk. Move pebble f to v, pebble b to u, and remove every other pebble on W.

  Cost: the number of unoccupied vertices on W.

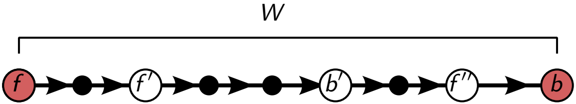
Legal moves

- **Forward move:** a forward pebble at u moves on an edge $u \to v$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.
- **Backward move:** a backward pebble at u moves on an edge $v \to u$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.
- **Flip move:** Let f be a forward pebble at u, let b be a backward pebble at v, and let W be a $u \to v$ walk. Move pebble f to v, pebble b to u, and remove every other pebble on W.

  Cost: the number of unoccupied vertices on W.



Legal moves

- **Forward move:** a forward pebble at u moves on an edge $u \to v$ to v.

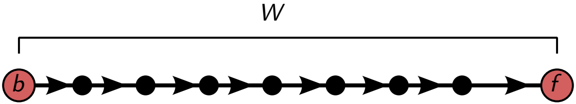
  Cost: 0 if v was already occupied, 1 otherwise.

- **Backward move:** a backward pebble at u moves on an edge $v \to u$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.

- **Flip move:** Let f be a forward pebble at u, let b be a backward pebble at v, and let W be a $u \to v$ walk. Move pebble f to v, pebble b to u, and remove every other pebble on W.

  Cost: the number of unoccupied vertices on W.



Legal moves

- **Forward move:** a forward pebble at u moves on an edge $u \to v$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.
- **Backward move:** a backward pebble at u moves on an edge $v \to u$ to v.

  Cost: 0 if v was already occupied, 1 otherwise.
- **Flip move:** Let f be a forward pebble at u, let b be a backward pebble at v, and let W be a $u \to v$ walk. Move pebble f to v, pebble b to u, and remove every other pebble on W.

  Cost: the number of unoccupied vertices on W.

Slight generalization: we allow the forward/backward moves on arbitrary $u \to v$ walks, not only on edges (and define the costs appropriately).

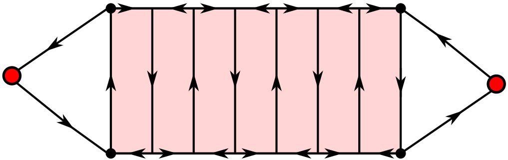
Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

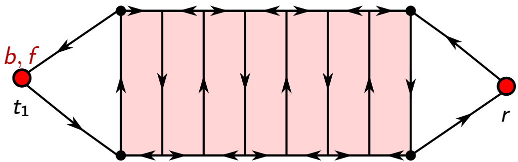
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

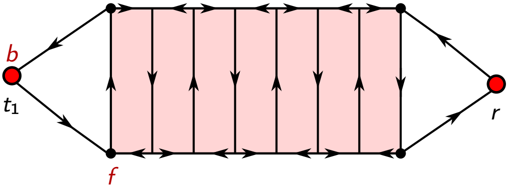
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

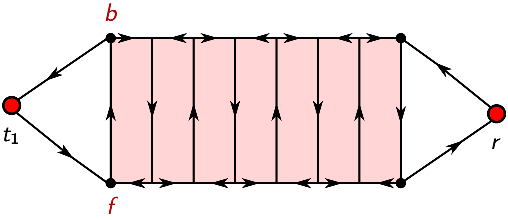
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

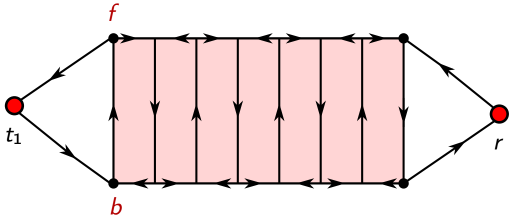
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

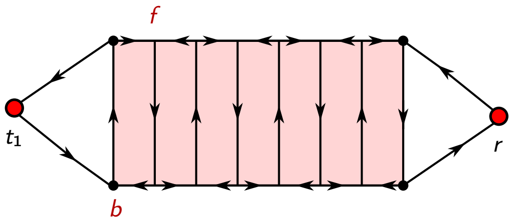
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

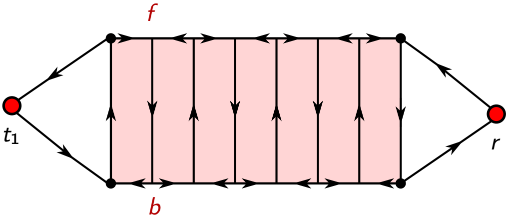
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

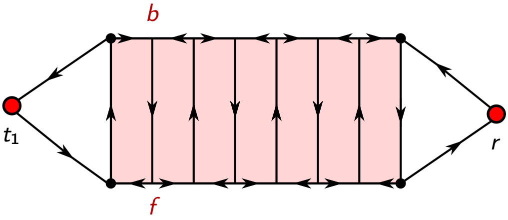
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

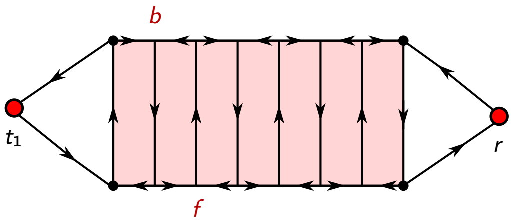
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

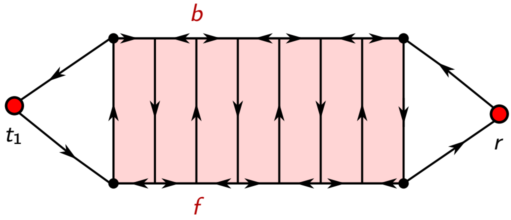
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

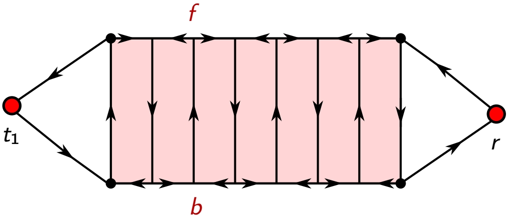
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

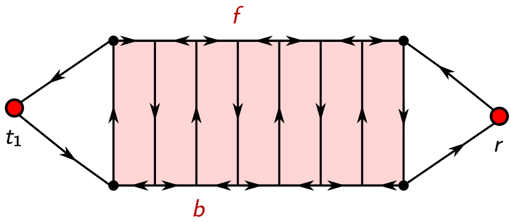
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

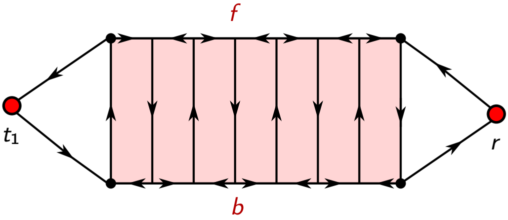
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

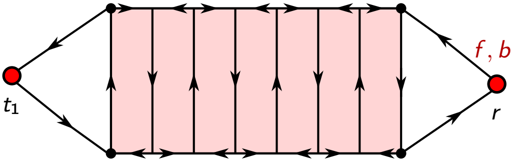
However, it is not true that the number of moves in a solution is bounded:



Bounding the number of moves

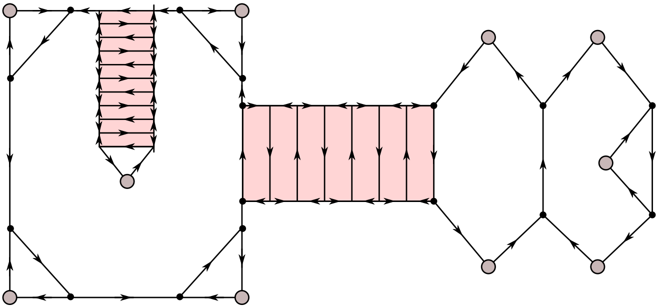
- Bound somehow the number of moves in an optimum solution.
- Argue that the moves form a planar graph with treewidth $O(\sqrt{k})$.
- Use standard treewidth techniques to find the best possible way this planar graph can appear.

However, it is not true that the number of moves in a solution is bounded:



Optimum solutions

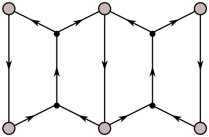
Closely looking at the $n^{O(k)}$ algorithm of [Feldman and Ruhl 2006] shows that an optimum solution consists of directed paths and "bidirectional strips":

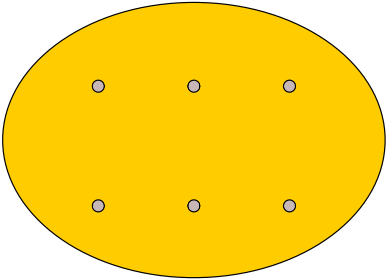


With some work, we can bound the number paths/strips by $O(k)$.

Algorithm

[Ignore the bidirectional strips for simplicity]

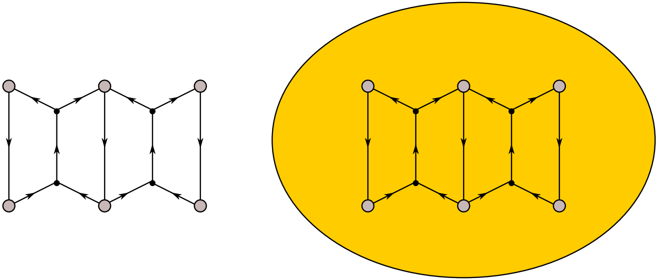




- We guess the topology of the solution ($2^{O(k \log k)}$ possibilities).
- As the number of moves is $O(k)$ and they form a planar graph, treewidth of the topology is $O(\sqrt{k})$.
- We can find the best realization of this topology (matching the location of the terminals) in time $n^{O(\sqrt{k})}$.

Algorithm

[Ignore the bidirectional strips for simplicity]



- We guess the topology of the solution ($2^{O(k \log k)}$ possibilities).
- As the number of moves is $O(k)$ and they form a planar graph, treewidth of the topology is $O(\sqrt{k})$.
- We can find the best realization of this topology (matching the location of the terminals) in time $n^{O(\sqrt{k})}$.

Lower bound:
The hardness result

Tight lower bounds

Theorem [Chen et al. 2004]

Assuming ETH, there is no $f(k) \cdot n^{o(k)}$ algorithm for k-Clique for any computable function f.

[ETH: n-variable 3SAT cannot be solved in time $2^{o(n)}$.]

Tight lower bounds

Theorem [Chen et al. 2004]

Assuming ETH, there is no $f(k) \cdot n^{o(k)}$ algorithm for k-Clique for any computable function f.

[ETH: n-variable 3SAT cannot be solved in time $2^{o(n)}$.]

Transfering to other problems:

k-Clique (x, k)	$\Rightarrow$	Problem A (x', k^2)
$f(k) \cdot n^{o(k)}$ algorithm	$\Leftarrow$	$f(k) \cdot n^{o(\sqrt{k})}$ algorithm

Bottom line:
To rule out $f(k) \cdot n^{o(\sqrt{k})}$ algorithms, we need a parameterized reduction that blows up the parameter at most quadratically.

Grid Tiling

Grid Tiling

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- Horizontal neighbors agree in the first component.
- Vertical neighbors agree in the second component.

(1,1) (1,3) (4,2)	(1,5) (4,1) (3,5)	(1,1) (4,2) (3,3)
(2,2) (4,1)	(1,3) (2,1)	(2,2) (3,2)
(3,1) (3,2) (3,3)	(1,1) (3,1)	(3,2) (3,5)

$k = 3$, $D = 5$

Grid Tiling

Grid Tiling

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- Horizontal neighbors agree in the first component.
- Vertical neighbors agree in the second component.

(1,1) (1,3) (4,2)	(1,5) (4,1) (3,5)	(1,1) (4,2) (3,3)
(2,2) (4,1)	(1,3) (2,1)	(2,2) (3,2)
(3,1) (3,2) (3,3)	(1,1) (3,1)	(3,2) (3,5)

$k = 3,\ D = 5$

Grid Tiling

Grid Tiling

Input: A $k \times k$ matrix and a set of pairs $S_{i,j} \subseteq [D] \times [D]$ for each cell.

Find: A pair $s_{i,j} \in S_{i,j}$ for each cell such that

- Horizontal neighbors agree in the first component.
- Vertical neighbors agree in the second component.

Fact

There is a parameterized reduction from k-Clique to $k \times k$ Grid Tiling.

Consequence

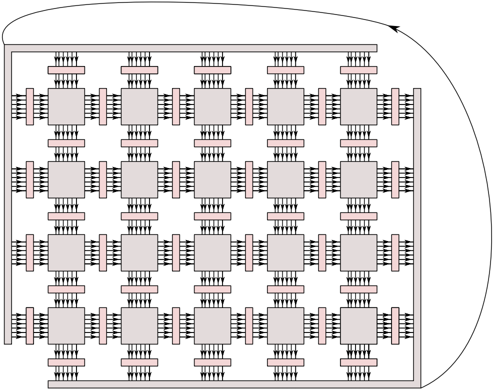
There is no $f(k)n^{o(k)}$ time algorithm for $k \times k$ Grid Tiling (assuming ETH).

Lower bound

Theorem

Strongly Connected Steiner Subgraph has no $f(k) \cdot n^{o(\sqrt{k})}$ time algorithm on planar directed graphs (assuming ETH).

The proof is by reduction from Grid Tiling and complicated construction of gadgets (constant number of terminals per gadget).



Lower bound

Theorem

Strongly Connected Steiner Subgraph has no $f(k) \cdot n^{o(\sqrt{k})}$ time algorithm on planar directed graphs (assuming ETH).

The proof is by reduction from Grid Tiling and complicated construction of gadgets (constant number of terminals per gadget).

k-Clique
(x, k)

$\Downarrow$

$k \times k$ Grid Tiling

$\Downarrow$

Strongly Connected Steiner Subgraph with $O(k^2)$ terminals

An extension:
Directed Steiner Forest

Steiner Forest

Generalization of STRONGLY CONNECTED STEINER SUBGRAPH:

DIRECTED STEINER FOREST

Input: A directed graph G, pairs of vertices (s_1, t_1), ..., (s_k, t_k).

Find: A subgraph F of G such that there is an $s_i \to t_i$ path in F for every $1 \leq i \leq k$.

Goal: Minimize the total weight of F.

Steiner Forest

Generalization of STRONGLY CONNECTED STEINER SUBGRAPH:

DIRECTED STEINER FOREST

Input: A directed graph G, pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$.

Find: A subgraph F of G such that there is an $s_i \to t_i$ path in F for every $1 \le i \le k$.

Goal: Minimize the total weight of F.

Theorem [Feldman and Ruhl 2006]

DIRECTED STEINER FOREST can be solved in time $n^{O(k)}$.

However, for DIRECTED STEINER FOREST $n^{O(k)}$ is best possible even on planar graphs:

Theorem

There is no $f(k)n^{o(k)}$ time algorithm for DIRECTED STEINER FOREST on planar graphs, unless ETH fails.

Summary

- On general graphs, the $n^{O(k)}$ algorithm of [Feldman and Ruhl 2006] for Strongly Connected Steiner Subgraph is essentially best possible (assuming ETH).
- On planar graphs, we can improve the running time to $f(k)n^{O(\sqrt{k})}$, but this is essentially best possible (assuming ETH).
  - Upper bound: massaging the problem into finding a graph of treewidth $O(\sqrt{k})$.
  - Lower bound: delicate reduction from Grid Tiling.
- Directed Steiner Forest: $n^{O(k)}$ algorithm of [Feldman and Ruhl 2006] is essentially best possible even on planar graphs (assuming ETH).