Parameterized Approximation Algorithms in Network Design

Andreas Emil Feldmann

joint work with:

Rajesh Chitnis, Pavel Dvořák, Dušan Knop, Pasin Manurangsi,
Tomáš Masařík, Tomáš Toufar, Pavel Veselý

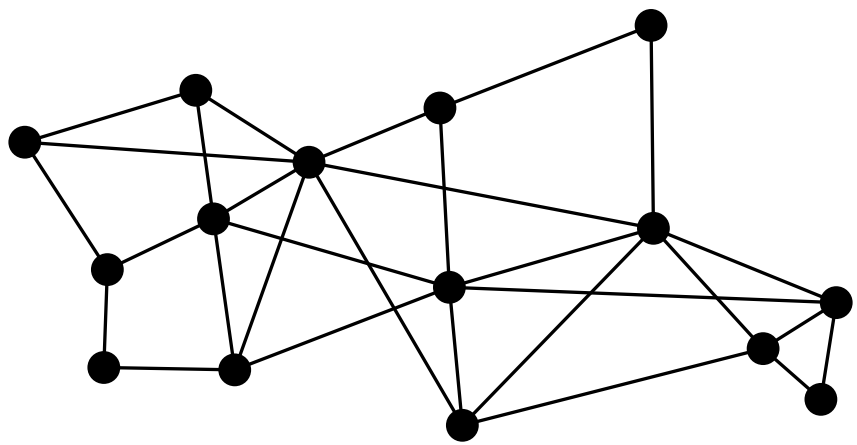
FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

KAM

Optimization problems

An *optimization problem* is given by

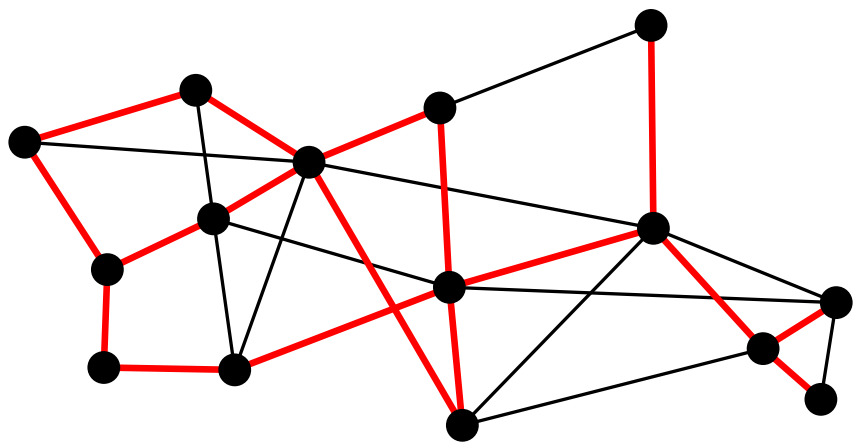
- **instances**, e.g., connected graphs with positive edge weights
- **feasible solutions**
- **cost function**



Optimization problems

An *optimization problem* is given by

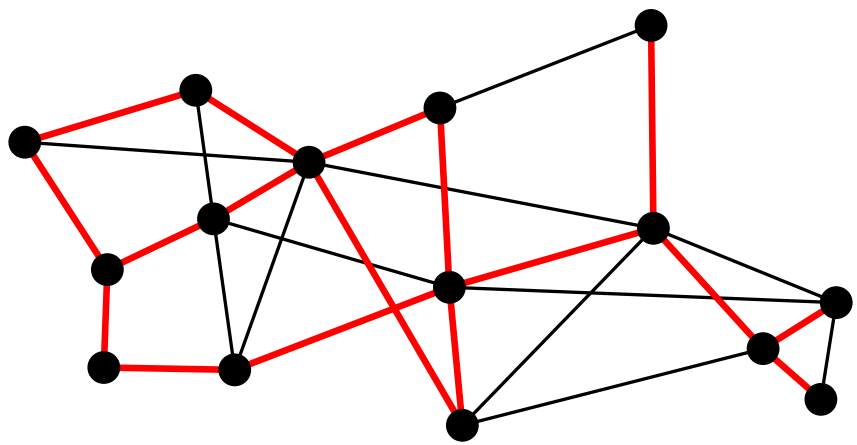
- **instances**, e.g., connected graphs with positive edge weights
- **feasible solutions**, e.g., spanning subgraphs
- **cost function**



Optimization problems

An *optimization problem* is given by

- **instances**, e.g., connected graphs with positive edge weights
- **feasible solutions**, e.g., spanning subgraphs
- **cost function**, e.g., total weight of a spanning subgraph

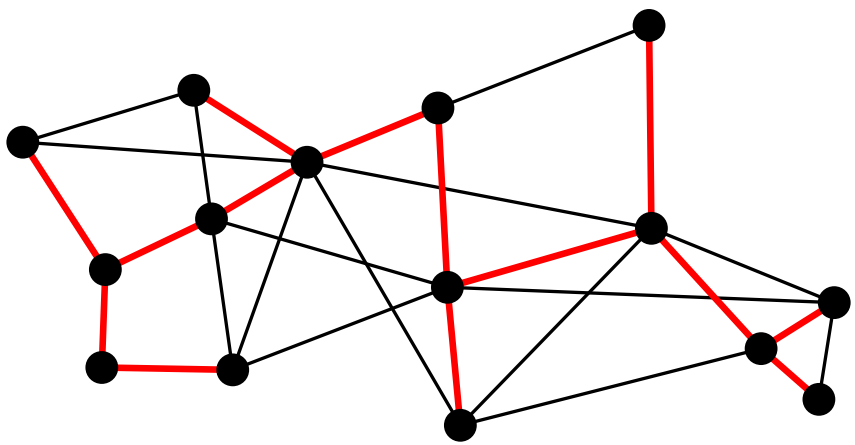


Optimization problems

An *optimization problem* is given by

- **instances**, e.g., connected graphs with positive edge weights
- **feasible solutions**, e.g., spanning subgraphs
- **cost function**, e.g., total weight of a spanning subgraph

Aim: given instance, compute **feasible solution of smallest cost**, e.g., Minimum Spanning Tree (MST)



Cobham-Edmonds thesis

A problem can be *feasibly solved* if there is an algorithm satisfying

1. **accuracy:**
   the algorithm always produces the optimum solution
2. **efficiency:**
   the algorithm runs in polynomial time in the size of the input

Cobham-Edmonds thesis

A problem can be *feasibly solved* if there is an algorithm satisfying

1. **accuracy:**
   the algorithm always produces the optimum solution
2. **efficiency:**
   the algorithm runs in polynomial time in the size of the input

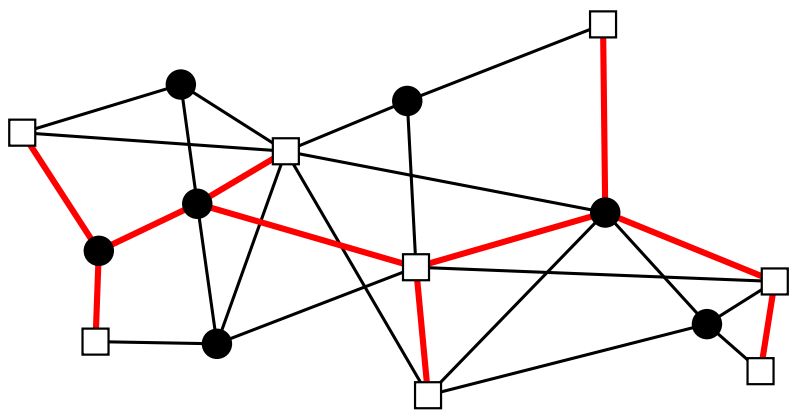
Theorem [Jarník '30]

The MST problem can be **feasibly solved**.

NP-hard problems

STEINER TREE (ST) problem

- instances: connected graphs $G = (V, E)$ with positive edge weights and **set $R \subseteq V$ of terminals** ($V \setminus R$ are called *Steiner vertices*)
- feasible solutions: connected subgraphs $T \subseteq G$ containing **all terminals in R**
- cost function: total weight of subgraph
  $\Rightarrow$ T is **Steiner tree**



NP-hard problems

Steiner Tree (ST) problem

- instances: connected graphs $G = (V, E)$ with positive edge weights and **set $R \subseteq V$ of terminals** ($V \setminus R$ are called *Steiner vertices*)
- feasible solutions: connected subgraphs $T \subseteq G$ containing **all terminals in R**
- cost function: total weight of subgraph
  $\Rightarrow$ T is **Steiner tree**

Theorem [Karp '72]

The ST problem is **NP-hard**, i.e., it cannot be feasibly solved, unless P=NP.

NP-hard problems

Steiner Tree (ST) problem

- instances: connected graphs $G = (V, E)$ with positive edge weights and **set $R \subseteq V$ of terminals** ($V \setminus R$ are called *Steiner vertices*)
- feasible solutions: connected subgraphs $T \subseteq G$ containing **all terminals in R**
- cost function: total weight of subgraph
  $\Rightarrow$ T is **Steiner tree**

Theorem [Karp '72]

The ST problem is **NP-hard**, i.e., it cannot be feasibly solved, unless P=NP.

Solution: give up at least one of **accuracy** or **efficiency**

Less accuracy: approximations

in practice: heuristics
hope: high quality output

Less accuracy: approximations

in practice: heuristics
hope: high quality output

"trust is good, verification is better"
$\rightarrow$ proof of guarantee

Less accuracy: approximations

in practice: heuristics	"trust is good, verification is better"
hope: high quality output	$\rightarrow$ proof of guarantee

Definitions

An α-*approximation* is a solution with cost at most α times the optimum.

Polynomial time α-*approximation algorithm*: compute α-approximation in polynomial time.

Polynomial time approximation scheme (PTAS): for any constant $\varepsilon > 0$, compute $(1 + \varepsilon)$-approximation in polynomial time (e.g. $n^{O(1/\varepsilon)}$).

Less accuracy: approximations

in practice: heuristics	"trust is good, verification is better"
hope: high quality output	$\rightarrow$ proof of guarantee

Definitions

An α-*approximation* is a solution with cost at most α times the optimum.

Polynomial time α-approximation algorithm: compute α-approximation in polynomial time.

Polynomial time approximation scheme (PTAS): for any constant $\varepsilon > 0$, compute $(1+\varepsilon)$-approximation in polynomial time (e.g. $n^{O(1/\varepsilon)}$).

Theorem [Byrka, Grandoni, Rothvoss, Sanità '13]

The ST problem has a **polynomial time** $(\ln(4)+\varepsilon)$-**approximation** algorithm for any constant $\varepsilon > 0$.

Less accuracy: approximations

in practice: heuristics	"trust is good, verification is better"
hope: high quality output	$\rightarrow$ proof of guarantee

Definitions

An α-*approximation* is a solution with cost at most α times the optimum.

Polynomial time α-approximation algorithm: compute α-approximation in polynomial time.

Polynomial time approximation scheme (PTAS): for any constant $\varepsilon > 0$, compute $(1+\varepsilon)$-approximation in polynomial time (e.g. $n^{O(1/\varepsilon)}$).

Theorem [Byrka, Grandoni, Rothvoss, Sanità '13]

The ST problem has a **polynomial time** $(\ln(4)+\varepsilon)$-**approximation** algorithm for any constant $\varepsilon > 0$.

Theorem [Chlebík, Chlebíková '02]

The ST problem is **APX-hard**, i.e., no PTAS, unless P=NP.

Less efficiency: parametrization

in practice: IP/SAT solvers
hope: fast on most inputs

Less efficiency: parametrization

in practice: IP/SAT solvers
hope: fast on most inputs

"trust is good, verification is better"
$\rightarrow$ isolate exponential runtime

Less efficiency: parametrization

in practice: IP/SAT solvers
hope: fast on most inputs

"trust is good, verification is better"
$\rightarrow$ isolate exponential runtime

Definitions

Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n

Less efficiency: parametrization

in practice: IP/SAT solvers
hope: fast on most inputs

"trust is good, verification is better"
$\rightarrow$ isolate exponential runtime

Definitions

Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n
XP algorithm: runtime $f(k)n^{g(k)}$, for functions f, g indep of n

Less efficiency: parametrization

in practice: IP/SAT solvers — "trust is good, verification is better"
hope: fast on most inputs — $\rightarrow$ isolate exponential runtime

Definitions

Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n
XP algorithm: runtime $f(k)n^{g(k)}$, for functions f, g indep of n

Examples for the ST problem

parameter	algorithm	
treewidth τ of input graph	$2^{O(\tau)}n^{O(1)}$	[Cygan et al. '11]

Less efficiency: parametrization

in practice: IP/SAT solvers — "trust is good, verification is better"
hope: fast on most inputs — $\rightarrow$ isolate exponential runtime

Definitions

Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n
XP algorithm: runtime $f(k)n^{g(k)}$, for functions f, g indep of n

Examples for the ST problem

parameter	algorithm	
treewidth τ of input graph	$2^{O(\tau)}n^{O(1)}$	[Cygan et al. '11]
nr. of terminals $\lvert R\rvert$	$2^{O(\lvert R\rvert)}n^{O(1)}$	[Dreyfus, Wagner '71]

Less efficiency: parametrization

in practice: IP/SAT solvers — "trust is good, verification is better"
hope: fast on most inputs — $\rightarrow$ isolate exponential runtime

Definitions

Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n
XP algorithm: runtime $f(k)n^{g(k)}$, for functions f, g indep of n

Examples for the ST problem

parameter	algorithm	
treewidth τ of input graph	$2^{O(\tau)}n^{O(1)}$	[Cygan et al. '11]
nr. of terminals $\lvert R\rvert$	$2^{O(\lvert R\rvert)}n^{O(1)}$	[Dreyfus, Wagner '71]
nr. of Steiner vertices $\lvert V \setminus R\rvert$	$2^{O(\lvert V\setminus R\rvert)}n^{O(1)}$	[folklore]

Less efficiency: parametrization

in practice: IP/SAT solvers — "trust is good, verification is better"
hope: fast on most inputs — $\rightarrow$ isolate exponential runtime

Definitions

Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n
XP algorithm: runtime $f(k)n^{g(k)}$, for functions f, g indep of n

Examples for the ST problem

parameter	algorithm	
treewidth τ of input graph	$2^{O(\tau)}n^{O(1)}$	[Cygan et al. '11]
nr. of terminals $\lvert R\rvert$	$2^{O(\lvert R\rvert)}n^{O(1)}$	[Dreyfus, Wagner '71]
nr. of Steiner vertices $\lvert V \setminus R\rvert$	$2^{O(\lvert V\setminus R\rvert)}n^{O(1)}$	[folklore]
nr. of Steiner vertices $\lvert V(T) \setminus R\rvert$ in the optimum solution T	$n^{O(\lvert V(T)\setminus R\rvert)}$	[folklore]

Less efficiency: parametrization

in practice: IP/SAT solvers — "trust is good, verification is better"
hope: fast on most inputs — $\rightarrow$ isolate exponential runtime

Definitions
Instance given with *parameter* $k \in \mathbb{N}$
FPT algorithm: runtime $f(k)n^{O(1)}$, for function f indep of n
XP algorithm: runtime $f(k)n^{g(k)}$, for functions f, g indep of n

Theorem [Mölle, Richter, Rossmanith '06]

The ST problem has an **FPT** algorithm for **parameter** $|R|$ with runtime $(2+\delta)^{|R|}n^{O(1)}$ for any constant $\delta > 0$.

Theorem [folklore]

The ST problem is **W[2]-hard** for **parameter** $|V(T) \setminus R|$, where T is an optimum solution, i.e., no FPT algorithm, unless FPT=W[2].

Less accuracy & efficiency: approximations & parameters

Theorem [Chlebík, Chlebíková '02]

The ST problem is **APX-hard**, i.e., there is no PTAS, unless P=NP.

Theorem [folklore]

The ST problem is **W[2]-hard** for **parameter** $|V(T) \setminus R|$, where T is an optimum Steiner tree, i.e., no FPT algorithm, unless FPT=W[2].

Less accuracy & efficiency: approximations & parameters

Theorem [Chlebík, Chlebíková '02]

The ST problem is **APX-hard**, i.e., there is no PTAS, unless P=NP.

Theorem [folklore]

The ST problem is **W[2]-hard** for **parameter** $|V(T) \setminus R|$, where T is an optimum Steiner tree, i.e., no FPT algorithm, unless FPT=W[2].

Definitions for parameter k

Parameterized α-approximation algorithm: compute α-approximation in $f(k)n^{O(1)}$ time for some function f.

Parameterized approximation scheme (PAS): for any constant $\varepsilon > 0$, compute $(1+\varepsilon)$-approximation in $f(k)n^{O(1)}$ time for some function f.

Less accuracy & efficiency: approximations & parameters

Theorem [Chlebík, Chlebíková '02]

The ST problem is **APX-hard**, i.e., there is no PTAS, unless P=NP.

Theorem [folklore]

The ST problem is **W[2]-hard** for **parameter** $|V(T) \setminus R|$, where T is an optimum Steiner tree, i.e., no FPT algorithm, unless FPT=W[2].

Definitions for parameter k

Parameterized α-approximation algorithm: compute α-approximation in $f(k)n^{O(1)}$ time for some function f.

Parameterized approximation scheme (PAS): for any constant $\varepsilon > 0$, compute $(1+\varepsilon)$-approximation in $f(k)n^{O(1)}$ time for some function f.

Theorem [Dvořák, F, Knop, Masařík, Toufar, Veselý '18]

The ST problem has a **PAS** for **parameter** $k = |V(T) \setminus R|$, where T is an optimum Steiner tree, with runtime $2^{O(k^2/\varepsilon^4)}n^{O(1)}$

Technique: Preprocessing

Preprocessing algorithm

in practice: data reduction

1. remove “easy parts” to obtain “core”
2. solve “core” (IP/SAT solvers or heuristics)

Preprocessing algorithm

in practice: data reduction

1. remove "easy parts" to obtain "core"
2. solve "core" (IP/SAT solvers or heuristics)

Definition [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

An α-*approximate preprocessing algorithm* $\mathcal{A}$ for parameter k is a pair of **polynomial time** algorithms:

- *reduction alg:* given instance (I, k), compute new instance (I', k')
- *lifting alg:* given β-approximation to (I', k'), compute $\alpha\beta$-approximation to (I, k)

Preprocessing algorithm

in practice: data reduction

1. remove "easy parts" to obtain "core"
2. solve "core" (IP/SAT solvers or heuristics)

Definition [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

An α-*approximate preprocessing algorithm* $\mathcal{A}$ for parameter k is a pair of **polynomial time** algorithms:

- *reduction alg:* given instance (I, k), compute new instance (I', k')
- *lifting alg:* given β-approximation to (I', k'), compute $\alpha\beta$-approximation to (I, k)

If $|I'| + k' \leq f(k)$ for some function f, then $\mathcal{A}$ is an α-*approximate kernel*,

Preprocessing algorithm

in practice: data reduction

1. remove "easy parts" to obtain "core"
2. solve "core" (IP/SAT solvers or heuristics)

Definition [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

An α-*approximate preprocessing algorithm* $\mathcal{A}$ for parameter k is a pair of **polynomial time** algorithms:

- *reduction alg:* given instance (I, k), compute new instance (I', k')
- *lifting alg:* given β-approximation to (I', k'), compute $\alpha\beta$-approximation to (I, k)

If $|I'| + k' \leq f(k)$ for some function f, then $\mathcal{A}$ is an α-*approximate kernel*, and if additionally $\alpha = 1 + \varepsilon$ for any $\varepsilon > 0$, then $\mathcal{A}$ is a *polynomial-sized approximate kernelization scheme (PSAKS)*.

Preprocessing algorithm

in practice: data reduction

1. remove "easy parts" to obtain "core"
2. solve "core" (IP/SAT solvers or heuristics)

Definition [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

An α-*approximate preprocessing algorithm* $\mathcal{A}$ for parameter k is a pair of **polynomial time** algorithms:

- *reduction alg:* given instance (I, k), compute new instance (I', k')
- *lifting alg:* given β-approximation to (I', k'), compute $\alpha\beta$-approximation to (I, k)

If $|I'| + k' \leq f(k)$ for some function f, then $\mathcal{A}$ is an α-*approximate kernel*, and if additionally $\alpha = 1 + \varepsilon$ for any $\varepsilon > 0$, then $\mathcal{A}$ is a *polynomial-sized approximate kernelization scheme (PSAKS)*.

Lemma [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

A problem with parameter k has a **parametrized α-approximation algorithm** if and only if it admits an **α-approximate kernel**.

Reduction algorithm for Steiner Tree

In each step:

add a tree to the solution, which

1. has simple structure (can be found in polynomial time),
2. has small weight,
3. contains many terminals.

Reduction algorithm for Steiner Tree

In each step:

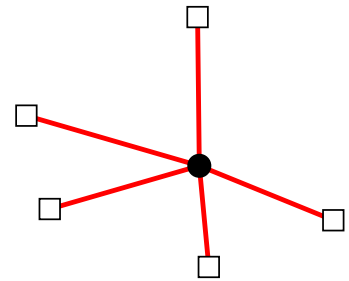
add a tree to the solution, which

1. has simple structure (can be found in polynomial time),
2. has small weight,
3. contains many terminals.

Definition

For star S with all leaves terminals, the *ratio* is

$$\frac{w(S)}{|V(S) \cap R| - 1}$$



Reduction algorithm for Steiner Tree

In each step:

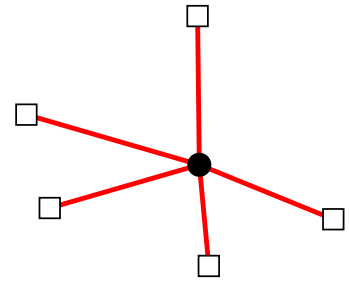
add a tree to the solution, which

1. has simple structure (can be found in polynomial time),
2. has small weight,
3. contains many terminals.

Definition

For star S with all leaves terminals, the *ratio* is

$$\frac{w(S)}{|V(S) \cap R| - 1}$$



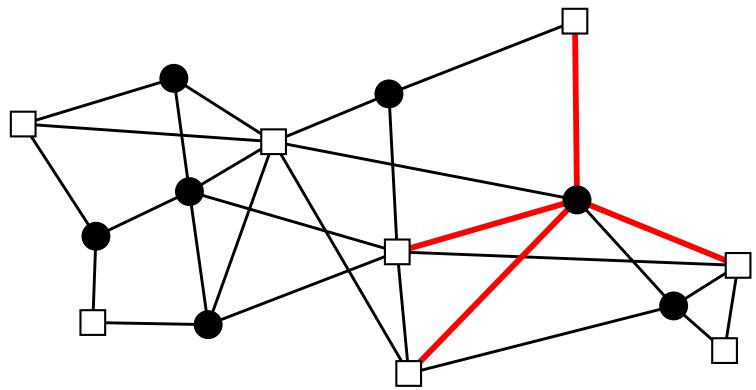
Lemma

A star of smallest ratio can be found in polynomial time.

A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

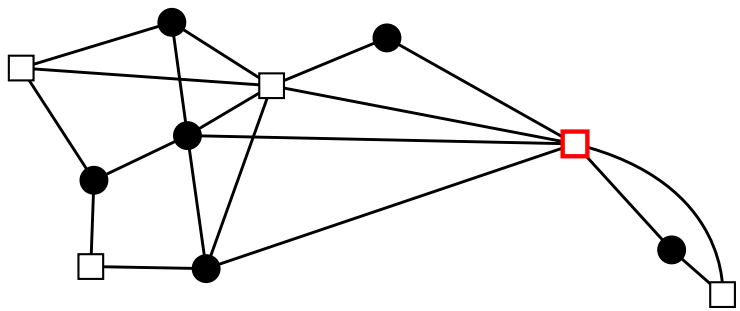
1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal



A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

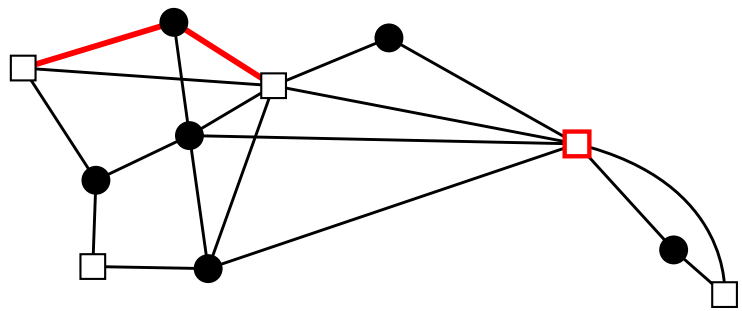
1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal



A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

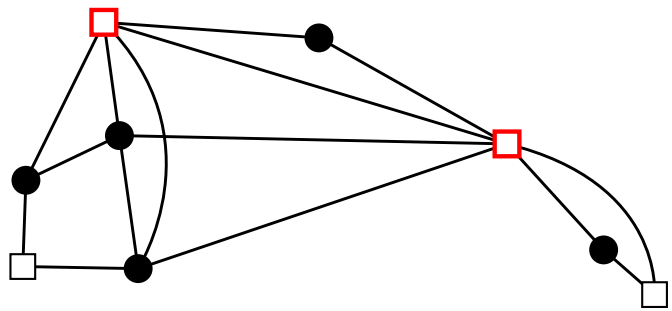
1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal



A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal

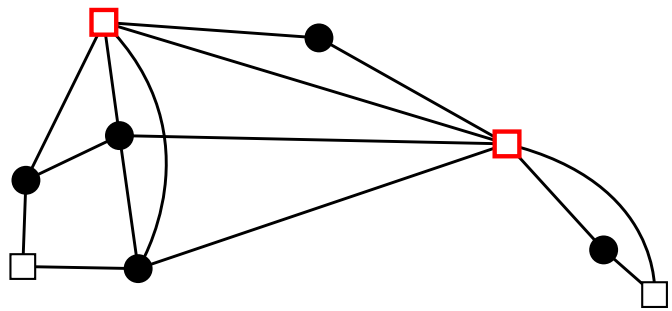


A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal

(note: not a kernel)

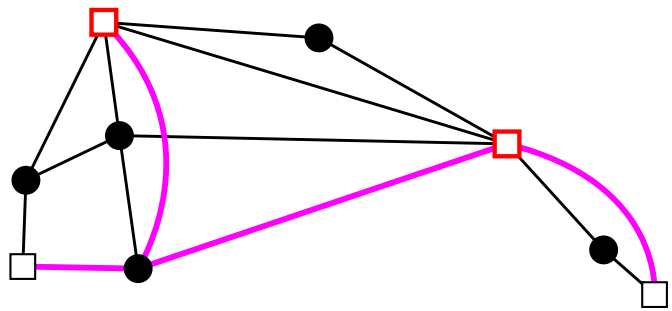


A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal

   (note: not a kernel)
2. solve remaining instance using FPT algorithm for parameter $|R|$

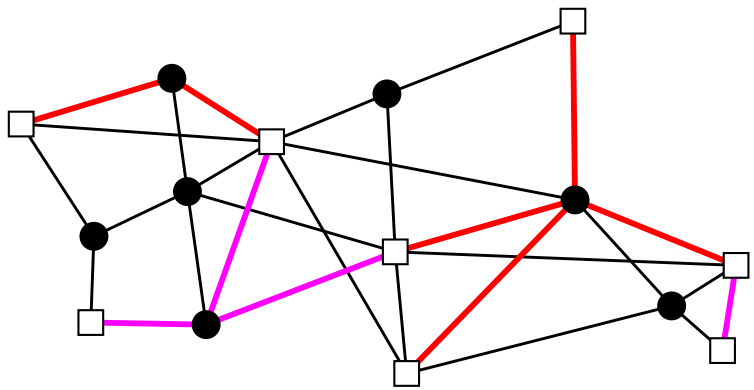


A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   1.1 find star S with smallest ratio (which exists if $|R| > k$)
   1.2 contract S and make resulting vertex a terminal

   (note: not a kernel)
2. solve remaining instance using FPT algorithm for parameter $|R|$
3. lifting: uncontract all stars in the solution computed by FPT alg

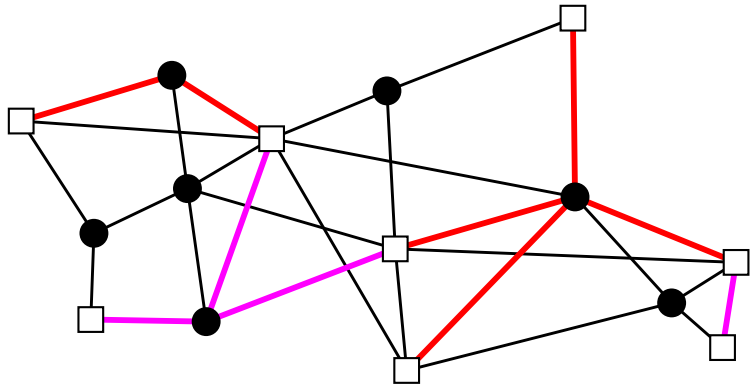


A parametrized approximation scheme

The algorithm for parameter $k = |V(T) \setminus R|$

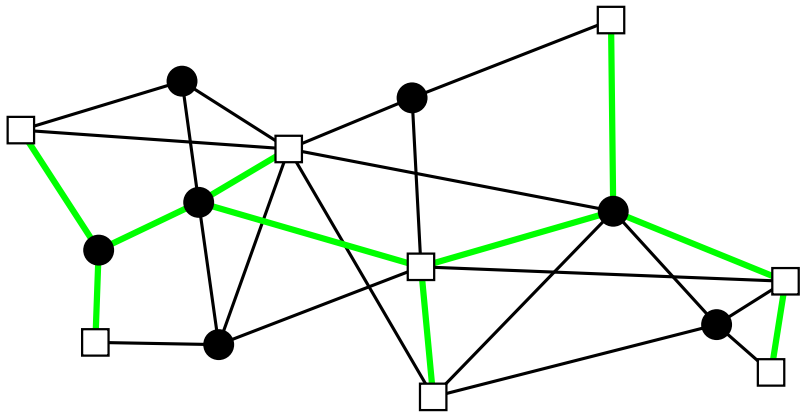
1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   - 1.1 find star S with smallest ratio (which exists if $|R| > k$)
   - 1.2 contract S and make resulting vertex a terminal

   (note: not a kernel)
2. solve remaining instance using FPT algorithm for parameter $|R|$
3. lifting: uncontract all stars in the solution computed by FPT alg

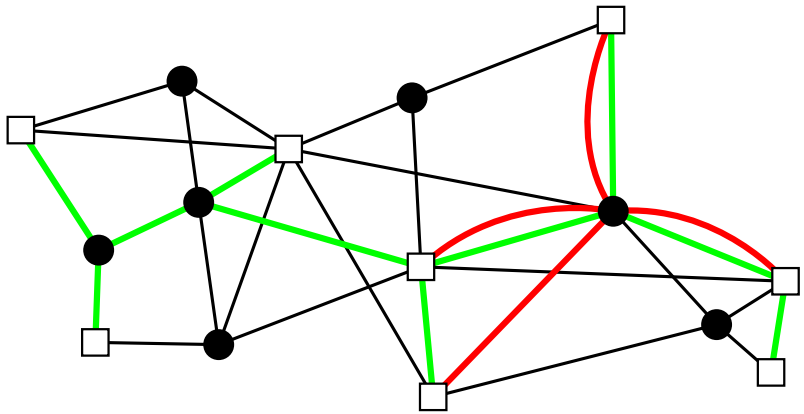


$\rightarrow$ runtime: $2^{O(k^2/\varepsilon^4)} n^{O(1)}$ as $|R| = O(k^2/\varepsilon^4)$ in step 2
due to FPT algorithm with runtime $(2+\delta)^{|R|} n^{O(1)}$
[Mölle, Richter, Rossmanith '06]

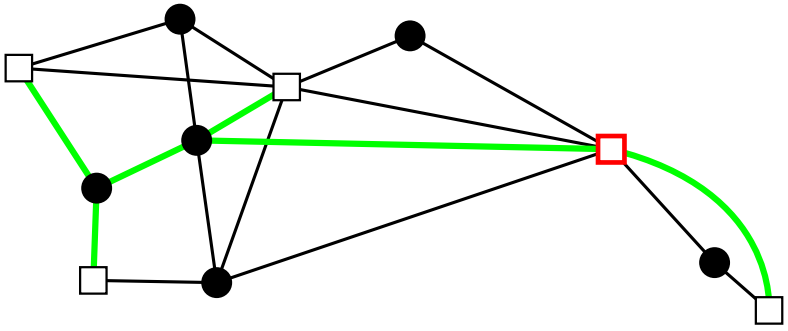
Charging stars against the optimum



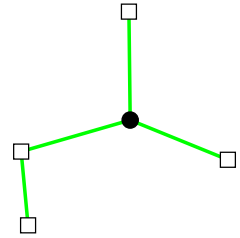
Charging stars against the optimum



Charging stars against the optimum

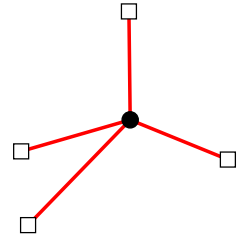


D_i:

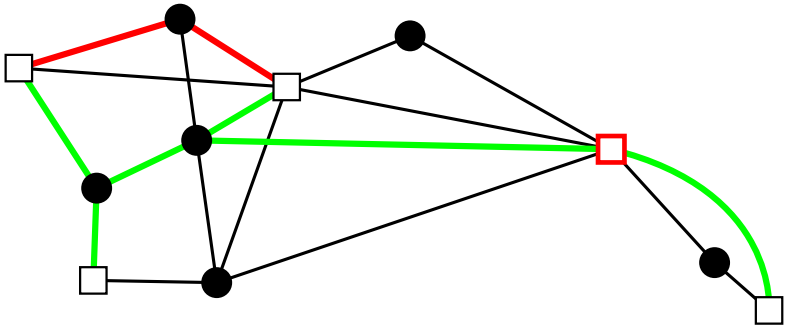


vs

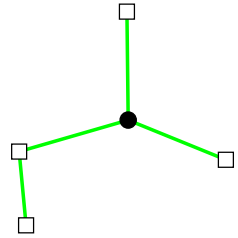
S_i:



Charging stars against the optimum

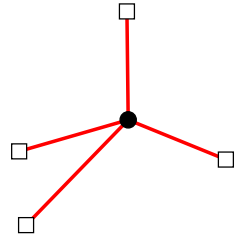


D_i:

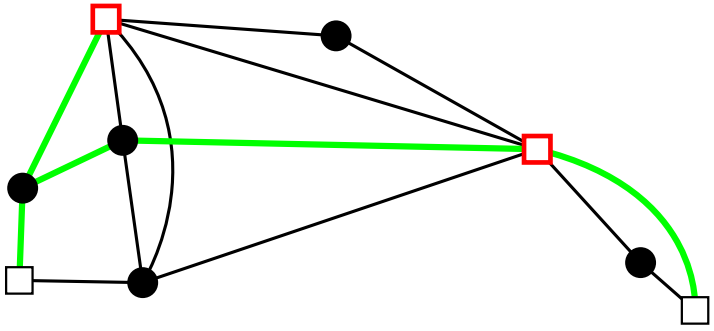


vs

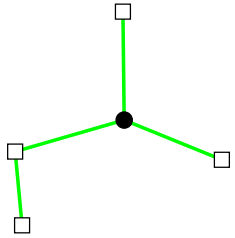
S_i:



Charging stars against the optimum



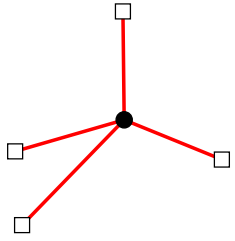
D_i:

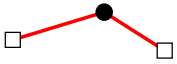


vs

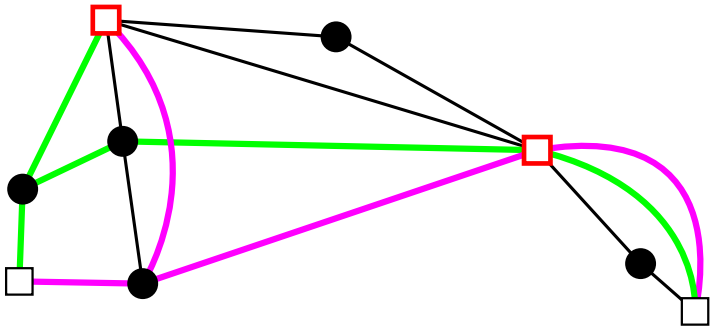
vs

S_i:

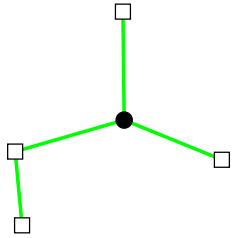




Charging stars against the optimum



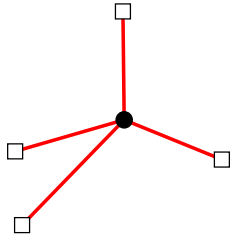
D_i:

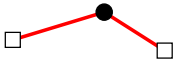


vs

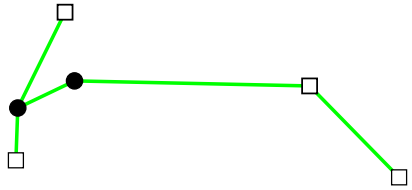
vs

S_i:

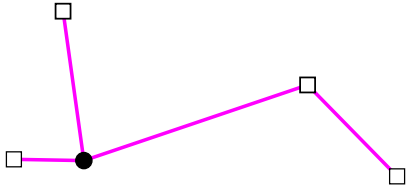




Charging stars against the optimum

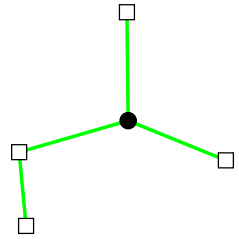


IV



$\Rightarrow (1+\varepsilon)$-approximation :-)

D_i:



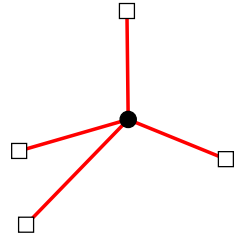
$\times(1+\varepsilon)$

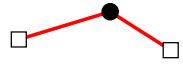
IV

$\times(1+\varepsilon)$

IV

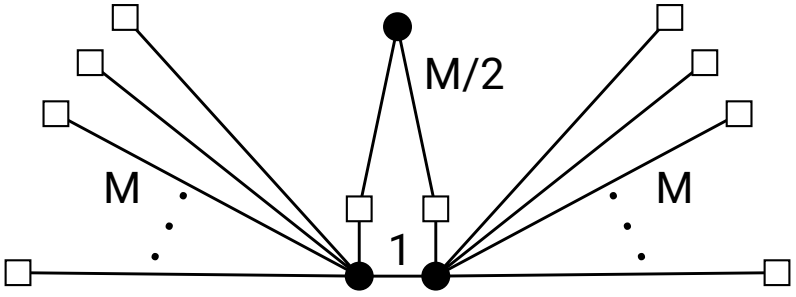
S_i:



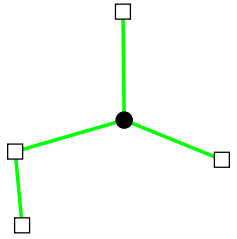


Charging stars against the optimum

A bad example: :-(



D_i:



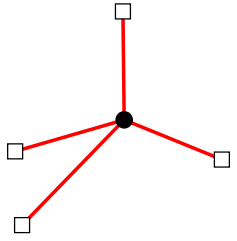
$\times(1+\varepsilon)$

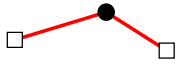
IV

$\times(1+\varepsilon)$

IV

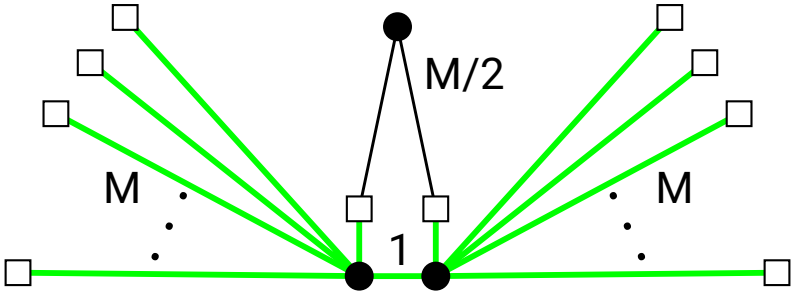
S_i:



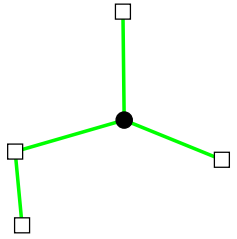


Charging stars against the optimum

A bad example: :-(



D_i:



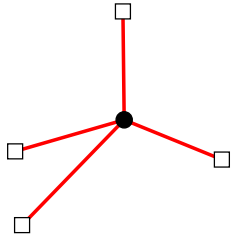
$\times(1+\varepsilon)$

IV

$\times(1+\varepsilon)$

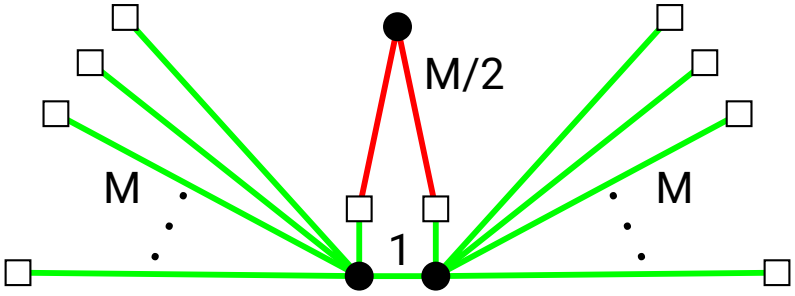
IV

S_i:

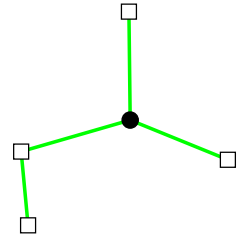


Charging stars against the optimum

A bad example: :-(



D_i:



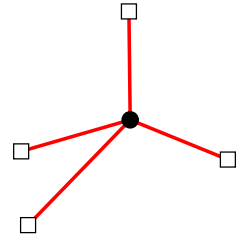
$\times(1+\varepsilon)$

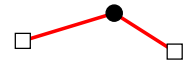
IV

$\times(1+\varepsilon)$

IV

S_i:





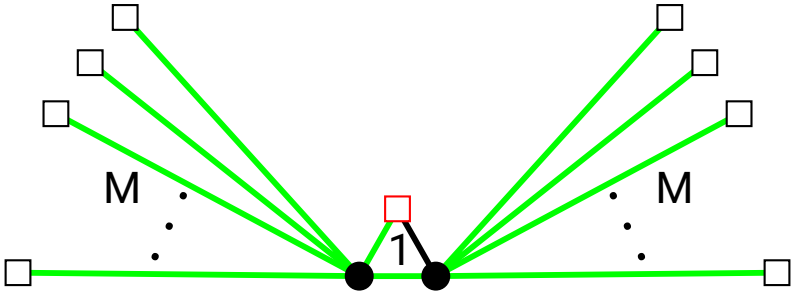
min ratio star:

$$\frac{w(S)}{|V(S)\cap R|-1} = \frac{2\cdot M/2}{2-1} = M$$

$$< \frac{\ell M}{\ell-1} \qquad \text{and} \qquad < \frac{\ell M+1}{\ell+1-1}$$

Charging stars against the optimum

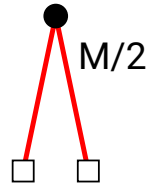
A bad example: :-(



D_i:

vs

S_i:



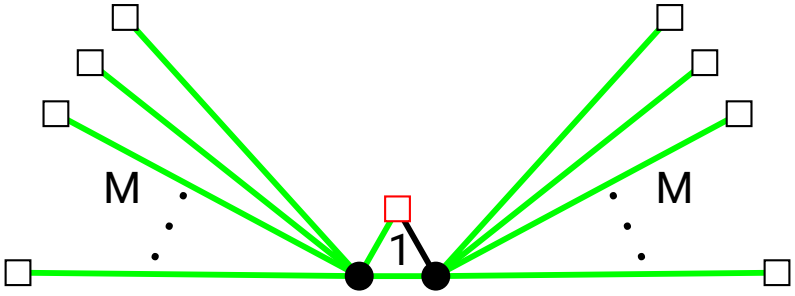
min ratio star:

$$\frac{w(S)}{|V(S) \cap R| - 1} = \frac{2 \cdot M/2}{2-1} = M$$

$$< \frac{\ell M}{\ell - 1} \qquad \text{and} \qquad < \frac{\ell M + 1}{\ell + 1 - 1}$$

Charging stars against the optimum

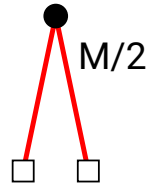
A bad example: :-(



D_i:

vs

S_i:



min ratio star:

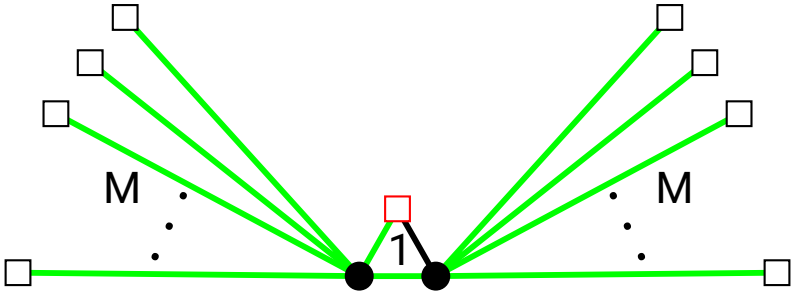
but: if $|R|$ is large we can charge against remaining optimum tree!

$$\frac{w(S)}{|V(S) \cap R| - 1} = \frac{2 \cdot M/2}{2-1} = M$$

$$< \frac{\ell M}{\ell - 1} \quad \text{and} \quad < \frac{\ell M + 1}{\ell + 1 - 1}$$

Charging stars against the optimum

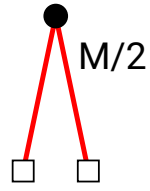
A bad example: :-(



D_i:

vs

S_i:



min ratio star:

$$\frac{w(S)}{|V(S) \cap R| - 1} = \frac{2 \cdot M/2}{2-1} = M$$

$$< \frac{\ell M}{\ell - 1} \quad \text{and} \quad < \frac{\ell M + 1}{\ell + 1 - 1}$$

but: if $|R|$ is large we can charge against remaining optimum tree!

Definition

A contracted star S_i is *bad* if $w(S_i) > (1+\varepsilon)w(D_i)$.
Otherwise it is *good*.

Charging bad stars against the optimum

Lemma

For the optimum Steiner tree T, parameter $k = |V(T) \setminus R|$, and threshold $\tau = \Theta(k^2/\varepsilon^4)$ we have

$$\sum_{\text{bad } S} w(S) \leq \varepsilon \cdot w(T)$$

Charging bad stars against the optimum

Lemma

For the optimum Steiner tree T, parameter $k = |V(T) \setminus R|$, and threshold $\tau = \Theta(k^2/\varepsilon^4)$ we have

$$\sum_{\text{bad } S} w(S) \leq \varepsilon \cdot w(T)$$

Theorem [Dvořák, F, Knop, Masařík, Toufar, Veselý '18]

The ST problem has a **PAS** for **parameter** $k = |V(T) \setminus R|$, where T is an optimum Steiner tree, with runtime $2^{O(k^2/\varepsilon^4)} n^{O(1)}$

Kernelization

A kernel for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   1.1 find star S with smallest ratio (which exists if $|R| > k$)
   1.2 contract S and make resulting vertex a terminal
2. **find a kernel for parameter** $|R|$
3. lifting: uncontract all stars in the solution computed by FPT alg

Kernelization

A kernel for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   1.1 find star S with smallest ratio (which exists if $|R| > k$)
   1.2 contract S and make resulting vertex a terminal
2. **find a kernel for parameter** $|R|$
3. lifting: uncontract all stars in the solution computed by FPT alg

Theorem [Dom, Lokshtanov, Saurabh '09]

The ST problem has **no polynomial-sized kernel** for **parameter** $|R|$, unless NP$\subseteq$coNP/poly.

Kernelization

A kernel for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   1.1 find star S with smallest ratio (which exists if $|R| > k$)
   1.2 contract S and make resulting vertex a terminal
2. **find a kernel for parameter** $|R|$
3. lifting: uncontract all stars in the solution computed by FPT alg

Theorem [Dom, Lokshtanov, Saurabh '09]

The ST problem has **no polynomial-sized kernel** for **parameter** $|R|$, unless NP$\subseteq$coNP/poly.

But using the Borchers & Du Theorem:

Theorem [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

The ST problem admits a **PSAKS** for **parameter** $|R|$ of size $|R|^{2^{O(1/\varepsilon)}}$.

Kernelization

A kernel for parameter $k = |V(T) \setminus R|$

1. reduction: as long as $|R| \geq \tau$ $(= \Theta(k^2/\varepsilon^4))$
   1.1 find star S with smallest ratio (which exists if $|R| > k$)
   1.2 contract S and make resulting vertex a terminal
2. **find a kernel for parameter** $|R|$
3. lifting: uncontract all stars in the solution computed by FPT alg

Theorem [Dom, Lokshtanov, Saurabh '09]

The ST problem has **no polynomial-sized kernel** for **parameter** $|R|$, unless NP$\subseteq$coNP/poly.

But using the Borchers & Du Theorem:

Theorem [Lokshtanov, Panolan, Ramanujan, Saurabh '17]

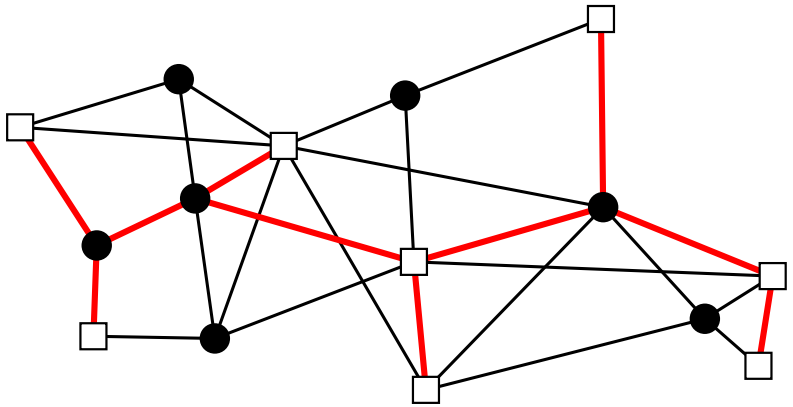
The ST problem admits a **PSAKS** for **parameter** $|R|$ of size $|R|^{2^{O(1/\varepsilon)}}$.

Theorem [Dvořák, F, Knop, Masařík, Toufar, Veselý '18]

The ST problem admits a **PSAKS** for **parameter** $k = |V(T) \setminus R|$, where T is an optimum Steiner tree, of size $k^{2^{O(1/\varepsilon)}}$.

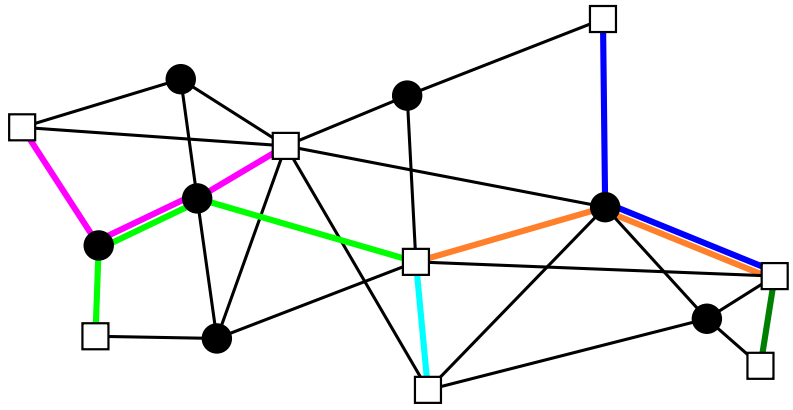
The Borchers & Du Theorem

Approximation of ST:
decomposition into small
instances is near-optimal



The Borchers & Du Theorem

Approximation of ST:
decomposition into small
instances is near-optimal

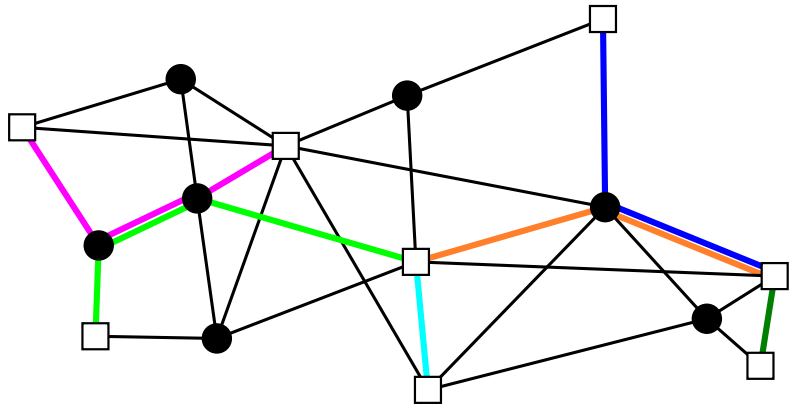


Definition

A *full-component* of a Steiner tree T is a subtree $T' \subseteq T$ for which all leaves are terminals. Let $R(T') = V(T') \cap R$ denote the terminals of T'.

The Borchers & Du Theorem

Approximation of ST: decomposition into small instances is near-optimal



Definition

A *full-component* of a Steiner tree T is a subtree $T' \subseteq T$ for which all leaves are terminals. Let $R(T') = V(T') \cap R$ denote the terminals of T'.

Theorem [Borchers, Du '97]

For every $\varepsilon > 0$ and optimum Steiner tree T there exists a set $\mathcal{T}$ of full-components of T, such that

- $|R(T')| \leq 2^{O(1/\varepsilon)}$ for every $T' \in \mathcal{T}$
- $\bigcup_{T' \in \mathcal{T}} T' = T$
- $\sum_{T' \in \mathcal{T}} w(T') \leq (1+\varepsilon) w(T)$

Algorithmic applications

Simplifying the input

- for every $R' \subseteq R$ for which $|R'| \leq 2^{O(1/\varepsilon)}$: compute optimum Steiner tree in time $2^{O(|R'|)}n^{O(1)}$ via FPT algorithm

Algorithmic applications

Simplifying the input

- for every $R' \subseteq R$ for which $|R'| \leq 2^{O(1/\varepsilon)}$: compute optimum Steiner tree in time $2^{O(|R'|)} n^{O(1)}$ via FPT algorithm

$\rightarrow$ by Thm: set of precomputed Steiner trees contains a subset $\mathcal{T}$, such that their union is a $(1+\varepsilon)$-approximation

Algorithmic applications

Simplifying the input

- for every $R' \subseteq R$ for which $|R'| \leq 2^{O(1/\varepsilon)}$: compute optimum Steiner tree in time $2^{O(|R'|)} n^{O(1)}$ via FPT algorithm

$\rightarrow$ by Thm: set of precomputed Steiner trees contains a subset $\mathcal{T}$, such that their union is a $(1+\varepsilon)$-approximation

$\rightarrow$ time: $\sum_{i=2}^{2^{O(1/\varepsilon)}} \binom{|R|}{i} \cdot 2^{O(i)} n^{O(1)} = |R|^{2^{O(1/\varepsilon)}} n^{O(1)}$

Algorithmic applications

Simplifying the input

- for every $R' \subseteq R$ for which $|R'| \leq 2^{O(1/\varepsilon)}$: compute optimum Steiner tree in time $2^{O(|R'|)} n^{O(1)}$ via FPT algorithm

$\rightarrow$ by Thm: set of precomputed Steiner trees contains a subset $\mathcal{T}$, such that their union is a $(1+\varepsilon)$-approximation

$\rightarrow$ time: $\sum_{i=2}^{2^{O(1/\varepsilon)}} \binom{|R|}{i} \cdot 2^{O(i)} n^{O(1)} = |R|^{2^{O(1/\varepsilon)}} n^{O(1)}$

Consequences

- find a good subset in polynomial time, e.g., via iterative rounding $\rightarrow$ $(\ln(4)+\varepsilon)$-approximation [Byrka, Grandoni, Rothvoss, Sanità '13]

Algorithmic applications

Simplifying the input

- for every $R' \subseteq R$ for which $|R'| \leq 2^{O(1/\varepsilon)}$: compute optimum Steiner tree in time $2^{O(|R'|)} n^{O(1)}$ via FPT algorithm

$\rightarrow$ by Thm: set of precomputed Steiner trees contains a subset $\mathcal{T}$, such that their union is a $(1+\varepsilon)$-approximation

$\rightarrow$ time: $\sum_{i=2}^{2^{O(1/\varepsilon)}} \binom{|R|}{i} \cdot 2^{O(i)} n^{O(1)} = |R|^{2^{O(1/\varepsilon)}} n^{O(1)}$

Consequences

- find a good subset in polynomial time, e.g., via iterative rounding $\rightarrow$ $(\ln(4)+\varepsilon)$-approximation [Byrka, Grandoni, Rothvoss, Sanità '13]
- sparsify union of precomputed Steiner trees: shortcut degree 2 Steiner vertices & encode edge lengths using $O(\log |R|/\varepsilon)$ bits $\rightarrow$ PSAKS of size $|R|^{2^{O(1/\varepsilon)}}$ [Lokshtanov, Panolan, Ramanujan, Saurabh '17] (vs: ST has no poly sized kernel for $|R|$)

Algorithmic applications

Simplifying the input

- for every $R' \subseteq R$ for which $|R'| \leq 2^{O(1/\varepsilon)}$: compute optimum Steiner tree in time $2^{O(|R'|)} n^{O(1)}$ via FPT algorithm

$\rightarrow$ by Thm: set of precomputed Steiner trees contains a subset $\mathcal{T}$, such that their union is a $(1+\varepsilon)$-approximation

$\rightarrow$ time: $\sum_{i=2}^{2^{O(1/\varepsilon)}} \binom{|R|}{i} \cdot 2^{O(i)} n^{O(1)} = |R|^{2^{O(1/\varepsilon)}} n^{O(1)}$

Consequences

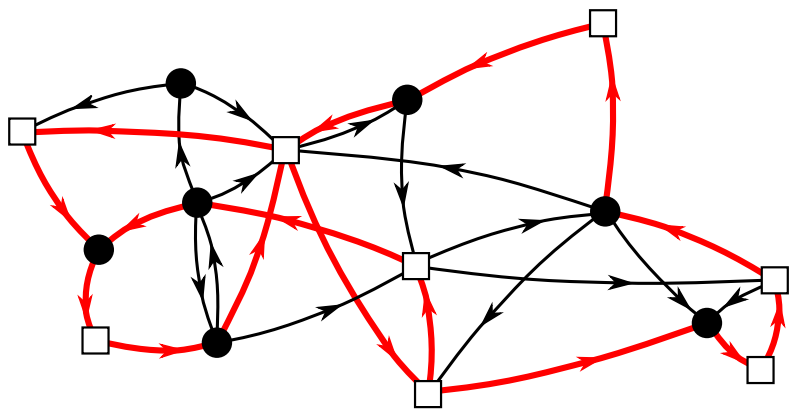
- find a good subset in polynomial time, e.g., via iterative rounding
  $\rightarrow$ $(\ln(4)+\varepsilon)$-approximation [Byrka, Grandoni, Rothvoss, Sanità '13]
- sparsify union of precomputed Steiner trees: shortcut degree 2 Steiner vertices & encode edge lengths using $O(\log |R|/\varepsilon)$ bits
  $\rightarrow$ PSAKS of size $|R|^{2^{O(1/\varepsilon)}}$ [Lokshtanov, Panolan, Ramanujan, Saurabh '17]
  (vs: ST has no poly sized kernel for $|R|$)
- preprocess instance to reduce number of terminals
  $\rightarrow$ PSAKS of size $k^{2^{O(1/\varepsilon)}}$ for $k = |V(T) \setminus R|$ and opt tree T
  [Dvořák, F, Knop, Masařík, Toufar, Veselý '18]
  (vs: ST is W[2]-hard for k)

Technique: Taking Two Easier Solutions

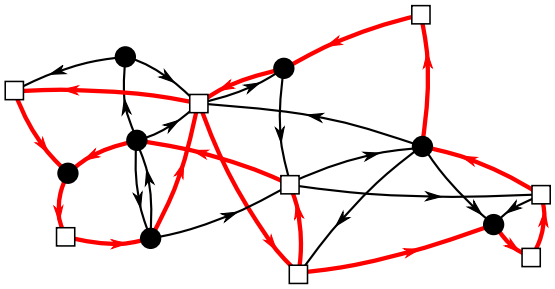
Strongly connected solutions

Strongly Connected Steiner Subgraph (SCSS) problem

- instances: **directed graphs** $G = (V, E)$ with positive edge weights and **terminal set** $R \subseteq V$
- feasible solutions: **strongly connected** subgraphs $N \subseteq G$ containing R
- cost function: total weight of directed subgraph
  $\Rightarrow N$ is **Steiner network**

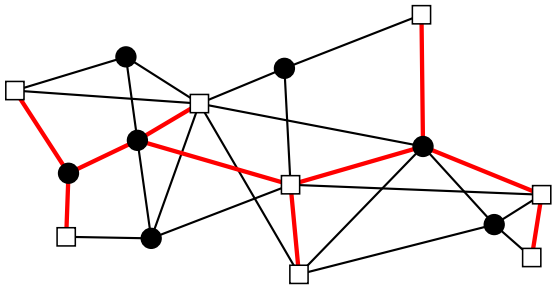


Parameterized vs approximation complexity



SCSS

harder than



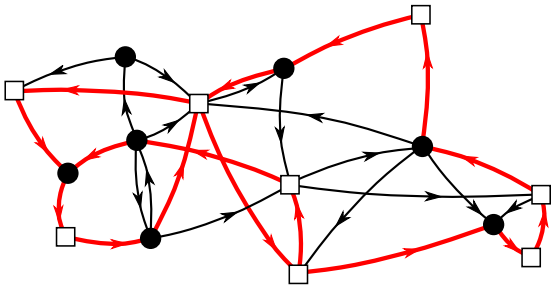
Steiner Tree

Parametrized complexity:

Theorem [Guo, Niedermeier, Suchý '11]

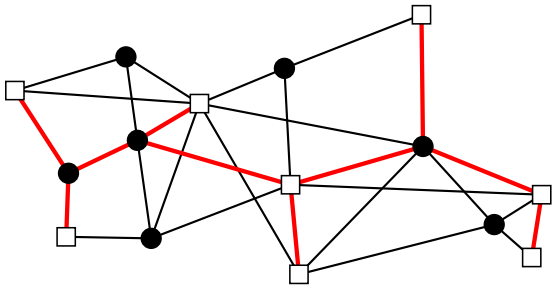
SCSS is **W[1]-hard** for **parameter** $|R|$, i.e., no FPT algorithm, unless FPT=W[1].

Parameterized vs approximation complexity



SCSS

harder than



Steiner Tree

Parametrized complexity:

Theorem [Guo, Niedermeier, Suchý '11]

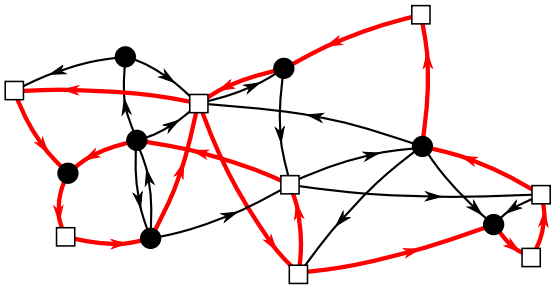
SCSS is **W[1]-hard** for **parameter** $|R|$, i.e., no FPT algorithm, unless FPT=W[1].

Approximation complexity:

Theorem [Halperin, Krauthgamer '03]

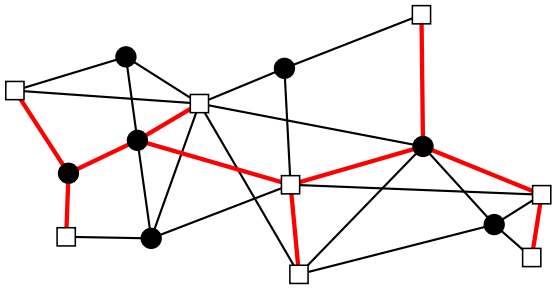
SCSS has **no polynomial time** $O(\log^{2-\varepsilon} n)$-**approximation** algorithm, unless NP $\subseteq$ ZTIME($n^{\text{polylog}(n)}$).

Parameterized vs approximation complexity



SCSS

harder than



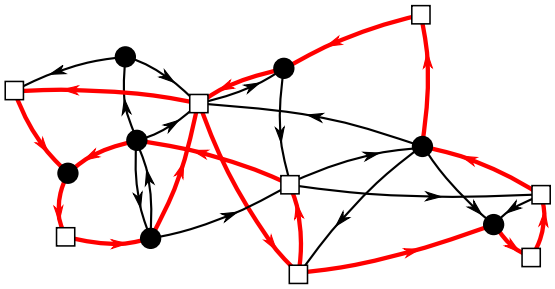
Steiner Tree

Parameterized approximation complexity:

Theorem [Chitnis, Hajiaghayi, Kortsarz '13]

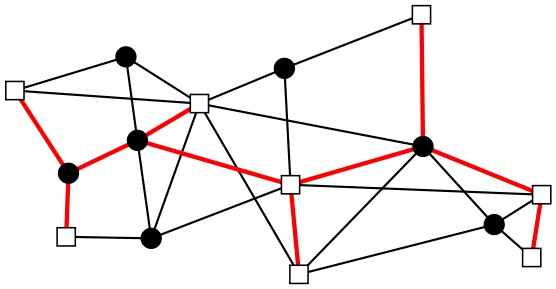
SCSS has a **parametrized** 2-**approximation** algorithm with **runtime** $2^{O(|R|)}n^{O(1)}$.

Parameterized vs approximation complexity



SCSS

harder than



Steiner Tree

Parameterized approximation complexity:

Theorem [Chitnis, Hajiaghayi, Kortsarz '13]

SCSS has a **parametrized** 2-**approximation** algorithm with **runtime** $2^{O(|R|)}n^{O(1)}$.

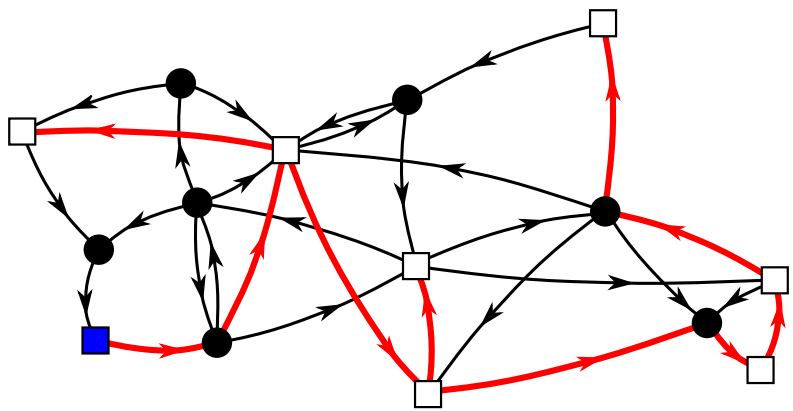
Theorem [Chitnis, F, Manurangsi '18]

SCSS has **no parametrized** $(2-\varepsilon)$-**approximation** algorithm with **runtime** $f(|R|)n^{O(1)}$, for any function f, under Gap-ETH.

Arborescences as solutions

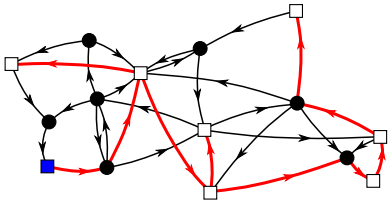
Directed Steiner Tree (DST) problem

- instances: **directed graphs** $G = (V, E)$ with positive edge weights, **terminal set** $R \subseteq V$, and **root terminal** $r \in R$
- feasible solutions: subgraphs $T \subseteq G$ containing $r \to t$ path for every $t \in R$
- cost function: total weight of directed subgraph
  $\Rightarrow T$ is **Steiner arborescence**



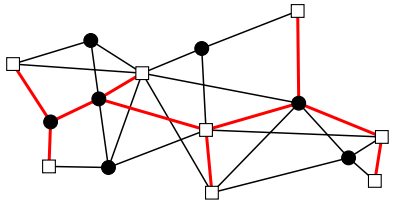
Directed vs undirected graphs

Parametrized complexity:



DST

similar to



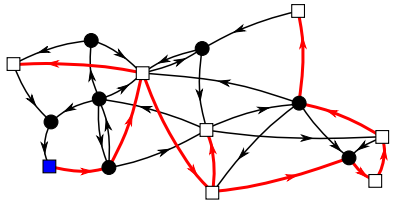
Steiner Tree

Theorem [Mölle, Richter, Rossmanith '06]

The DST problem has an **FPT** algorithm for **parameter** $|R|$ with runtime $(2+\delta)^{|R|} n^{O(1)}$ for any constant $\delta > 0$.

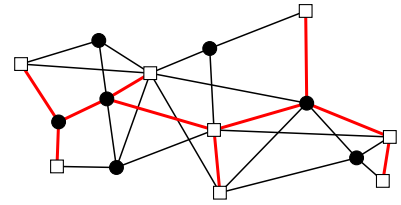
Directed vs undirected graphs

Parametrized complexity:



DST

similar to

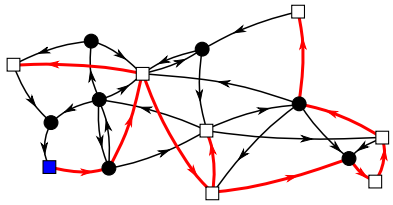


Steiner Tree

Theorem [Mölle, Richter, Rossmanith '06]

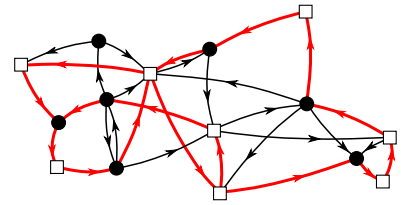
The DST problem has an **FPT** algorithm for **parameter** $|R|$ with runtime $(2+\delta)^{|R|} n^{O(1)}$ for any constant $\delta > 0$.

Approximation complexity:



DST

similar to



SCSS

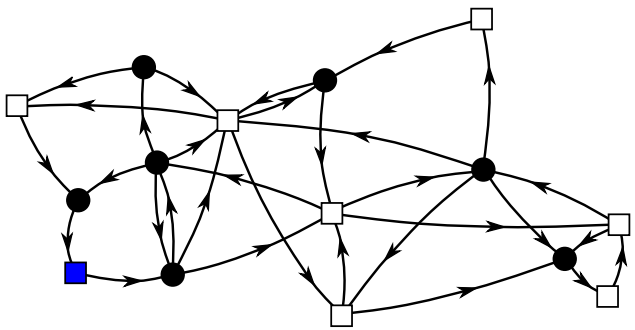
Theorem [Halperin, Krauthgamer '03]

DST has **no polynomial time** $O(\log^{2-\varepsilon} n)$-**approximation** algorithm, unless NP $\subseteq$ ZTIME($n^{\text{polylog}(n)}$).

Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

1. declare arbitrary terminal to be root r

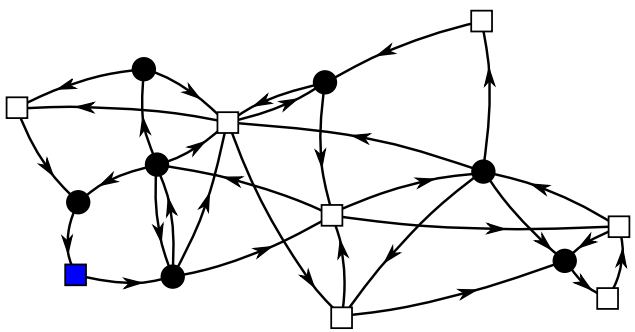


G with root $r \in R$

Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

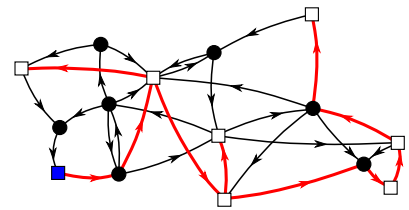
1. declare arbitrary terminal to be root r
2. compute DST solution T_{out} in G with $r \to t$ paths for all $t \in R$



G with root $r \in R$

$\longrightarrow$

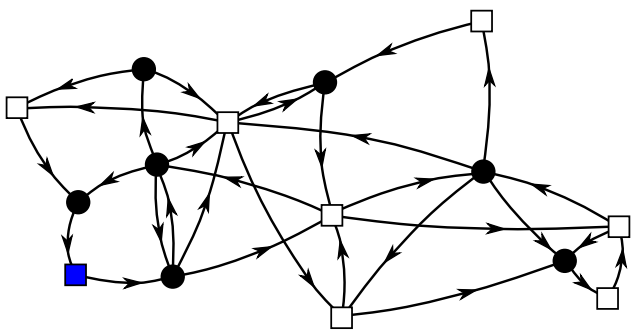
T_{out}



Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

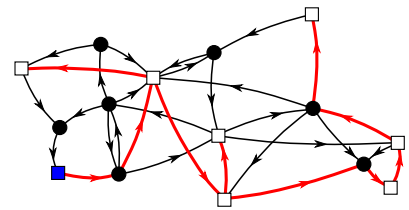
1. declare arbitrary terminal to be root r
2. compute DST solution T_{out} in G with $r \to t$ paths for all $t \in R$
3. compute DST solution T_{in} in G with $t \to r$ paths for all $t \in R$



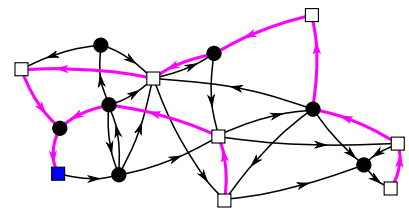
G with root $r \in R$

$\longrightarrow$

T_{out}



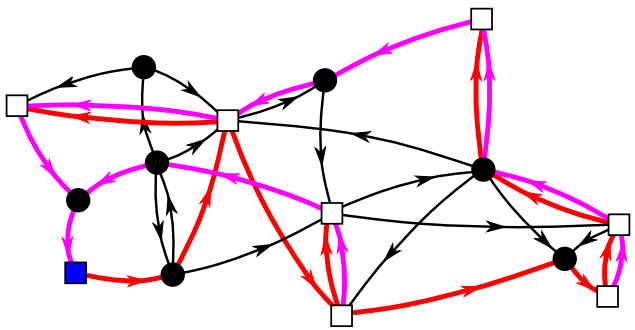
T_{in}



Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

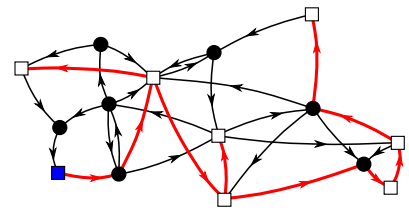
1. declare arbitrary terminal to be root r
2. compute DST solution T_{out} in G with $r \to t$ paths for all $t \in R$
3. compute DST solution T_{in} in G with $t \to r$ paths for all $t \in R$
4. output union N of T_{in} and T_{out}



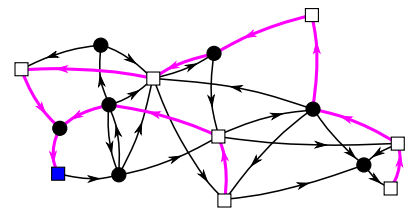
$N = T_{in} \cup T_{out}$

$\longleftarrow$

T_{out}



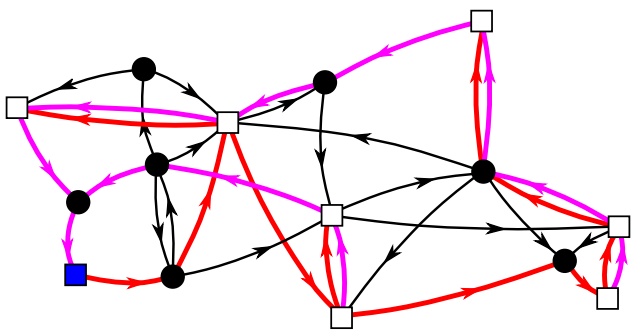
T_{in}



Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

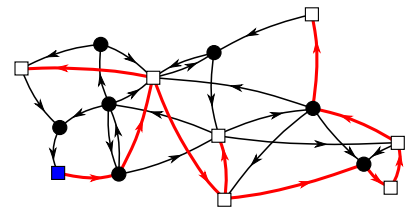
1. declare arbitrary terminal to be root r
2. compute DST solution T_{out} in G with $r \to t$ paths for all $t \in R$
3. compute DST solution T_{in} in G with $t \to r$ paths for all $t \in R$
4. output union N of T_{in} and T_{out}



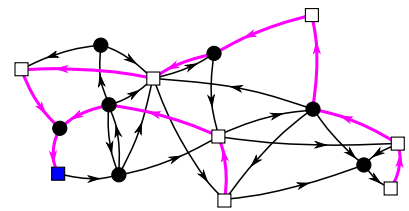
$N = T_{in} \cup T_{out}$

$\longleftarrow$

T_{out}



T_{in}



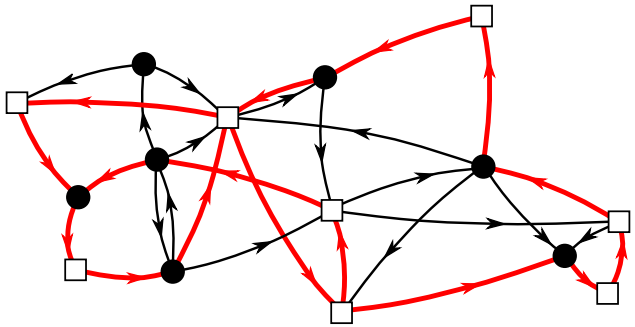
Approximation factor

$w(N) \leq w(T_{in}) + w(T_{out})$

Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

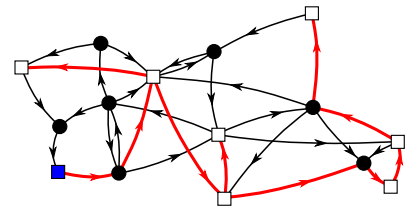
1. declare arbitrary terminal to be root r
2. compute DST solution T_{out} in G with $r \to t$ paths for all $t \in R$
3. compute DST solution T_{in} in G with $t \to r$ paths for all $t \in R$
4. output union N of T_{in} and T_{out}



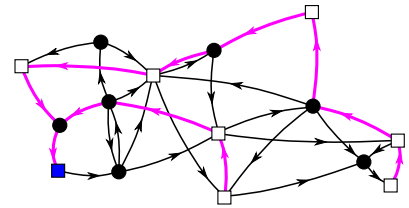
optimum solution N^*

$\geq$

T_{out}



T_{in}



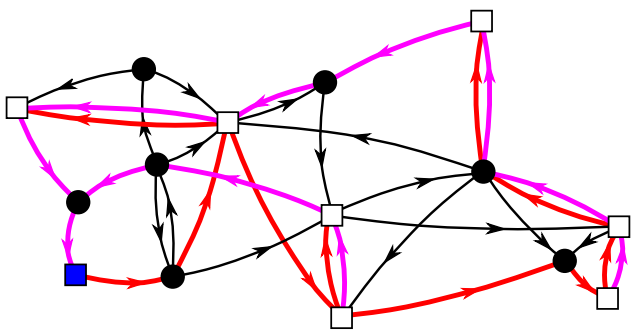
Approximation factor

$$w(N) \leq w(T_{in}) + w(T_{out}) \leq w(N^*) + w(N^*) = 2 \cdot w(N^*)$$

Approximating SCSS via two solutions to DST

Algorithm for directed graph G and terminal set R

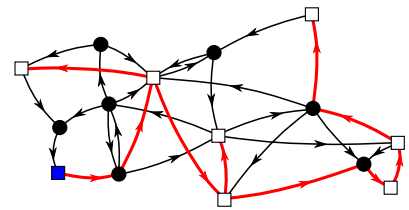
1. declare arbitrary terminal to be root r
2. compute DST solution T_{out} in G with $r \to t$ paths for all $t \in R$
3. compute DST solution T_{in} in G with $t \to r$ paths for all $t \in R$
4. output union N of T_{in} and T_{out}



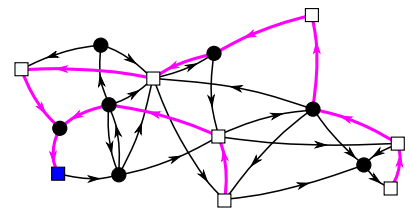
$N = T_{in} \cup T_{out}$

$\longleftarrow$

T_{out}



T_{in}



Theorem [Chitnis, Hajiaghayi, Kortsarz '13]

SCSS has a **parametrized** 2-**approximation** algorithm with **runtime** $2^{O(|R|)} n^{O(1)}$.

Technique: Reductions from Densest k-Subgraph

A tight lower bound for SCSS

Theorem [Guo, Niedermeier, Suchý '11]

SCSS is **W[1]-hard** for **parameter** $|R|$.

A tight lower bound for SCSS

Theorem [Guo, Niedermeier, Suchý '11]

SCSS is **W[1]-hard** for **parameter** $|R|$.

The Multicoloured Clique problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: clique on $v_1, \ldots, v_k \in V$ such that $v_i \in V_i$ for every $i \in [k]$
- parameter: k

A tight lower bound for SCSS

Theorem [Guo, Niedermeier, Suchý '11]

SCSS is **W[1]-hard** for **parameter** $|R|$.

The Multicoloured Clique problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: clique on $v_1, \ldots, v_k \in V$ such that $v_i \in V_i$ for every $i \in [k]$
- parameter: k

Reduction to SCSS

- reduce to unweighted graph
- parameter: $|R| = O(k^2)$ (one terminal for each pair $(i, j) \in \binom{k}{2}$)

$\Rightarrow$ no $f(|R|)n^{o(\sqrt{|R|})}$ time alg for SCSS,
since Clique has no $f(k)n^{o(k)}$ time alg under ETH,
i.e., assuming 3SAT cannot be solved in $2^{o(n)}$ time.

A tight lower bound for SCSS

Theorem [Guo, Niedermeier, Suchý '11]

SCSS has **no** $f(|R|)n^{o(|R|/\log|R|)}$ **time** algorithm for any function f, under ETH, i.e., assuming 3SAT cannot be solved in $2^{o(n)}$ time.

The Subgraph Isomorphism problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$, and graph H on k vertices
- find: $H' \subseteq G$ isomorphic to H s.t. $\forall i \in [k] : |V(H') \cap V_i| = 1$
- parameter: $p = |E(H)|$

A tight lower bound for SCSS

Theorem [Guo, Niedermeier, Suchý '11]

SCSS has **no** $f(|R|)n^{o(|R|/\log|R|)}$ **time** algorithm for any function f, under ETH, i.e., assuming 3SAT cannot be solved in $2^{o(n)}$ time.

The Subgraph Isomorphism problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$, and graph H on k vertices
- find: $H' \subseteq G$ isomorphic to H s.t. $\forall i \in [k] : |V(H') \cap V_i| = 1$
- parameter: $p = |E(H)|$

Reduction to SCSS: (almost) the same as from Clique

- reduce to unweighted graph
- parameter: $|R| = O(p)$ (one terminal for each edge of H)

$\Rightarrow$ under ETH: no $f(|R|)n^{o(|R|/\log|R|)}$ time alg for SCSS, since Subgraph Isomorphism has no $f(p)n^{o(p/\log p)}$ time alg [Marx '07] (open: get rid of $\log p$)

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has **no parametrized** $(2-\varepsilon)$**-approximation** algorithm with **runtime** $f(|R|)n^{O(1)}$, for any function f, under Gap-ETH, i.e, assuming no $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has **no parametrized** $(2-\varepsilon)$**-approximation** algorithm with **runtime** $f(|R|)n^{O(1)}$, for any function f, under Gap-ETH, i.e, assuming no $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

The Densest k-Subgraph (DkS) problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: subgraph $H \subseteq G$ with max nr of edges s.t. $\forall i \in [k] : |V(H) \cap V_i| = 1$
- parameter: $q = \binom{k}{2}$, i.e., max possible nr of edges

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has **no parametrized** $(2-\varepsilon)$**-approximation** algorithm with **runtime** $f(|R|)n^{O(1)}$, for any function f, under Gap-ETH, i.e, assuming no $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

The Densest k-Subgraph (DkS) problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: subgraph $H \subseteq G$ with max nr of edges s.t. $\forall i \in [k] : |V(H) \cap V_i| = 1$
- parameter: $q = \binom{k}{2}$, i.e., max possible nr of edges

also known as: Binary CSP or Label Cover

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has **no parametrized** $(2-\varepsilon)$**-approximation** algorithm with **runtime** $f(|R|)n^{O(1)}$, for any function f, under Gap-ETH, i.e, assuming no $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

The Densest k-Subgraph (DkS) problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: subgraph $H \subseteq G$ with max nr of edges s.t. $\forall i \in [k] : |V(H) \cap V_i| = 1$
- parameter: $q = \binom{k}{2}$, i.e., max possible nr of edges

Theorem [Dinur, Manurangsi '18]

Under Gap-ETH, no $f(q)n^{O(1)}$ time algorithm can distinguish between the following two cases for DkS:

- (completeness) there is H with $|E(H)| = q$
- (soundness) every H has $|E(H)| \leq q/k^{1-o(1)}$

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has **no parametrized** $(2-\varepsilon)$-**approximation** algorithm with **runtime** $f(|R|)n^{O(1)}$, for any function f, under Gap-ETH, i.e, assuming no $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

The Densest k-Subgraph (DkS) problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: subgraph $H \subseteq G$ with max nr of edges s.t. $\forall i \in [k] : |V(H) \cap V_i| = 1$
- parameter: $q = \binom{k}{2}$, i.e., max possible nr of edges

Reduction to SCSS: similar as from Subgraph Isomorphism

- reduce to **weighted** graph
- parameter: $|R| = O(q)$ (one terminal for each potential edge of H)

$\Rightarrow$ under Gap-ETH: no parameterized $2-\varepsilon$-approximation alg with runtime $f(|R|)n^{O(1)}$ for SCSS

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has no parametrized $(2-\varepsilon)$-approximation algorithm with **runtime** $f(|R|)n^{o(|R|)}$, for any function f, under

randomized Gap-ETH, i.e, assuming no **randomized** $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

The Densest k-Subgraph (DkS) problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: subgraph $H \subseteq G$ with max nr of edges s.t. $\forall i \in [k] : |V(H) \cap V_i| = 1$
- parameter: $q = \binom{k}{2}$, i.e., max possible nr of edges

Reduction to SCSS: similar as from Subgraph Isomorphism

- reduce to **weighted** graph
- parameter: $|R| = O(q)$ (one terminal for each potential edge of H)

$\Rightarrow$ under Gap-ETH: no parameterized $2-\varepsilon$-approximation alg with runtime $f(|R|)n^{O(1)}$ for SCSS

A tight lower bound for SCSS

Theorem [Chitnis, F, Manurangsi '18]

SCSS has no parametrized $(2-\varepsilon)$-approximation algorithm with **runtime** $f(|R|)n^{o(|R|)}$, for any function f, under

randomized Gap-ETH, i.e, assuming no **randomized** $2^{o(n)}$ time algorithm can distinguish if all or at most a $1-\delta$ fraction of 3SAT clauses are satisfiable, for some constant $\delta > 0$.

The Densest k-Subgraph (DkS) problem

- given: graph $G = (V, E)$ with vertex partition $V_1, \ldots, V_k$
- find: subgraph $H \subseteq G$ with max nr of edges s.t. $\forall i \in [k] : |V(H) \cap V_i| = 1$
- parameter: $q = \binom{k}{2}$, i.e., max possible nr of edges

Theorem [Manurangsi '20]

Under randomized Gap-ETH, for **any constant** $\alpha > 0$ **no** $f(q)n^{o(q)}$ **time** algorithm can distinguish between the following two cases for DkS:

- (completeness) there is H with $|E(H)| = q$
- (soundness) every H has $|E(H)| \leq \alpha q$

Outlook

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. KNAPSACK, ...
2. constant factors: e.g. STEINER TREE, ...
3. (poly-)logarithmic factors: e.g. SET COVER, ...
4. polynomial factors: e.g. CLIQUE, ...

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. KNAPSACK, ...
2. constant factors: e.g. STEINER TREE, ...
3. (poly-)logarithmic factors: e.g. SET COVER, ...
4. polynomial factors: e.g. CLIQUE, ...

Tight parameterized approximation factors:

(where a parameter admits a better factor than any poly time algorithm)

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. KNAPSACK, ...
2. constant factors: e.g. STEINER TREE, ...
3. (poly-)logarithmic factors: e.g. SET COVER, ...
4. polynomial factors: e.g. CLIQUE, ...

Tight parameterized approximation factors:

(where a parameter admits a better factor than any poly time algorithm)

1. approx schemes: e.g. STEINER TREE param $|V(T) \setminus R|$, ...

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. Knapsack, ...
2. constant factors: e.g. Steiner Tree, ...
3. (poly-)logarithmic factors: e.g. Set Cover, ...
4. polynomial factors: e.g. Clique, ...

Tight parameterized approximation factors:

(where a parameter admits a better factor than any poly time algorithm)

1. approx schemes: e.g. Steiner Tree param $|V(T) \setminus R|$, ...
2. constant factors: (i) SCSS param $|R|$, (ii) Directed Odd Cycle Transversal param solution size

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. Knapsack, ...
2. constant factors: e.g. Steiner Tree, ...
3. (poly-)logarithmic factors: e.g. Set Cover, ...
4. polynomial factors: e.g. Clique, ...

Tight parameterized approximation factors:

(where a parameter admits a better factor than any poly time algorithm)

1. approx schemes: e.g. Steiner Tree param $|V(T) \setminus R|$, ...
2. constant factors: (i) SCSS param $|R|$, (ii) Directed Odd Cycle Transversal param solution size
3. (poly-)logarithmic factors: ?

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. KNAPSACK, ...
2. constant factors: e.g. STEINER TREE, ...
3. (poly-)logarithmic factors: e.g. SET COVER, ...
4. polynomial factors: e.g. CLIQUE, ...

Tight parameterized approximation factors:

(where a parameter admits a better factor than any poly time algorithm)

1. approx schemes: e.g. STEINER TREE param $|V(T) \setminus R|$, ...
2. constant factors: (i) SCSS param $|R|$, (ii) DIRECTED ODD CYCLE TRANSVERSAL param solution size
3. (poly-)logarithmic factors: ?
4. polynomial factors: ?

Approximation factors

Tight polynomial time approximation factors:

1. approx schemes: e.g. Knapsack, ...
2. constant factors: e.g. Steiner Tree, ...
3. (poly-)logarithmic factors: e.g. Set Cover, ...
4. polynomial factors: e.g. Clique, ...

Tight parameterized approximation factors:

(where a parameter admits a better factor than any poly time algorithm)

1. approx schemes: e.g. Steiner Tree param $|V(T) \setminus R|$, ...
2. constant factors: (i) SCSS param $|R|$, (ii) Directed Odd Cycle Transversal param solution size
3. (poly-)logarithmic factors: ?
4. polynomial factors: ?
5. no reasonable factor: e.g. Dominating Set param solution size, ... (no $g(k)$-approx in $f(k)n^{o(k)}$ time for any g, f under ETH)

Running times

Tight parameterized runtimes:

1. sub-exponential $2^{O(\sqrt{k})}n^{O(1)}$: e.g. Planar Vertex Cover, ...
2. single exponential $2^{O(k)}n^{O(1)}$: e.g. Vertex Cover, ...
3. double exponential $2^{2^{O(k)}}n^{O(1)}$: e.g. Edge Clique Cover, ...
4. sub-exponential $n^{O(\sqrt{|R|})}$: e.g. Planar SCSS, ...
5. exponential $n^{O(|R|)}$: e.g. SCSS, ...
6. paraNP-hard: e.g. Vertex Colouring, ...

Running times

Tight parameterized runtimes:

1. sub-exponential $2^{O(\sqrt{k})}n^{O(1)}$: e.g. Planar Vertex Cover, ...
2. single exponential $2^{O(k)}n^{O(1)}$: e.g. Vertex Cover, ...
3. double exponential $2^{2^{O(k)}}n^{O(1)}$: e.g. Edge Clique Cover, ...
4. sub-exponential $n^{O(\sqrt{|R|})}$: e.g. Planar SCSS, ...
5. exponential $n^{O(|R|)}$: e.g. SCSS, ...
6. paraNP-hard: e.g. Vertex Colouring, ...

Tight parameterized approximation runtimes:

none ... (only some for algorithms with runtime exponential in n)

Running times

Tight parameterized runtimes:

1. sub-exponential $2^{O(\sqrt{k})}n^{O(1)}$: e.g. Planar Vertex Cover, ...
2. single exponential $2^{O(k)}n^{O(1)}$: e.g. Vertex Cover, ...
3. double exponential $2^{2^{O(k)}}n^{O(1)}$: e.g. Edge Clique Cover, ...
4. sub-exponential $n^{O(\sqrt{|R|})}$: e.g. Planar SCSS, ...
5. exponential $n^{O(|R|)}$: e.g. SCSS, ...
6. paraNP-hard: e.g. Vertex Colouring, ...

Tight parameterized approximation runtimes:

none ... (only some for algorithms with runtime exponential in n)

Example of open problem: SCSS

- 2-approximation in $2^{O(|R|)}n^{O(1)}$ time
- no $(2-\varepsilon)$-approximation in $f(|R|)n^{O(1)}$ time, under Gap-ETH

$\rightarrow$ can you show: no $2^{o(|R|)}n^{O(1)}$ time alg to compute 2-approximation?

Thanks for your attention!

Summary

- **preprocessing algorithm:** parametrized approximation scheme for ST with parameter $|V(T) \setminus R|$, where T is opt Steiner tree
- **take two easier solutions:** parametrized 2-approximation for SCSS with parameter $|R|$
- **reduction from Densest k-Subgraph:** no $(2-\varepsilon)$-approximation for SCSS with parameter $|R|$, under Gap-ETH
- **outlook:**
  1. tight approximation factors
  2. tight runtime bounds