Degree-Constrained Subgraph Problems: Hardness and Approximation

Omid Amini - *Max Planck (Germany)*
David Peleg - *Weizmann Inst. (Israel)*
Stéphane Perénnes - *CNRS (France)*
Ignasi Sau - CNRS (France) + UPC (Spain)
Saket Saurabh - *Univ. Bergen (Norway)*

Mascotte Project - INRIA/CNRS-I3S/UNSA - FRANCE
Applied Mathematics IV Department of UPC - SPAIN

IMFM Ljubljana - December 18th, 2008

Outline of the talk (ALGO/WAOA'08)

- Introduction / Preliminaries
- Problem 1
  - Definition + results
  - An approximation algorithm
- Problem 2
  - Definition + results
  - A hardness result
  - An approximation algorithm
- Problem 3
  - Definition + results
- Further research

Degree-Constrained Subgraph Problems

Broad family of problems

- A *typical* **DEGREE-CONSTRAINED SUBGRAPH PROBLEM**:

  Input:

  - a (*weighted* or *unweighted*) graph G, and
  - an integer d.

  Output:

  - a (*connected*) subgraph H of G,
  - satisfying some degree constraints ($\Delta(H) \leq d$ or $\delta(H) \geq d$),
  - and optimizing some parameter ($|V(H)|$ or $|E(H)|$).

- Several problems in this broad family are classical widely studied NP-hard problems.

- They have a number of applications in interconnection networks, routing algorithms, chemistry, ...

Broad family of problems

- A *typical* **DEGREE-CONSTRAINED SUBGRAPH PROBLEM**:

  Input:

  - a (*weighted* or *unweighted*) graph G, and
  - an integer d.

  Output:

  - a (*connected*) subgraph H of G,
  - satisfying some degree constraints ($\Delta(H) \leq d$ or $\delta(H) \geq d$),
  - and optimizing some parameter ($|V(H)|$ or $|E(H)|$).

- Several problems in this broad family are classical widely studied NP-hard problems.

- They have a number of applications in interconnection networks, routing algorithms, chemistry, ...

Broad family of problems

- A *typical* **DEGREE-CONSTRAINED SUBGRAPH PROBLEM**:

  Input:

  - a (*weighted* or *unweighted*) graph G, and
  - an integer d.

  Output:

  - a (*connected*) subgraph H of G,
  - satisfying some degree constraints ($\Delta(H) \leq d$ or $\delta(H) \geq d$),
  - and optimizing some parameter ($|V(H)|$ or $|E(H)|$).

- Several problems in this broad family are classical widely studied NP-hard problems.

- They have a number of applications in interconnection networks, routing algorithms, chemistry, ...

Broad family of problems

- A *typical* **DEGREE-CONSTRAINED SUBGRAPH PROBLEM**:

  Input:

  - a (*weighted* or *unweighted*) graph G, and
  - an integer d.

  Output:

  - a (*connected*) subgraph H of G,
  - satisfying some degree constraints ($\Delta(H) \leq d$ or $\delta(H) \geq d$),
  - and optimizing some parameter ($|V(H)|$ or $|E(H)|$).

- Several problems in this broad family are classical widely studied NP-hard problems.

- They have a number of applications in interconnection networks, routing algorithms, chemistry, ...

Broad family of problems

- A *typical* **DEGREE-CONSTRAINED SUBGRAPH PROBLEM**:

  Input:
  - a (*weighted* or *unweighted*) graph G, and
  - an integer d.

  Output:
  - a (*connected*) subgraph H of G,
  - satisfying some degree constraints ($\Delta(H) \leq d$ or $\delta(H) \geq d$),
  - and optimizing some parameter ($|V(H)|$ or $|E(H)|$).

- Several problems in this broad family are classical widely studied NP-hard problems.

- They have a number of applications in interconnection networks, routing algorithms, chemistry, ...

Preliminaries: approximation algorithms

- Given a (typically NP-hard) *minimization* problem Π, we say that ALG is an α-approximation algorithm for Π (with $\alpha \geq 1$) if for any instance I of Π,

$$ALG(I) \leq \alpha \cdot OPT(I).$$

- **Example**:

  MINIMUM VERTEX COVER

  Input: An undirected graph $G = (V, E)$.

  Output: A subset $S \subseteq V$ such that for each $\{u, v\} \in E$, at least one of u and v is in S, and such that $|S|$ is minimized.

- Approximation algorithm for MINIMUM VERTEX COVER:
  $\longrightarrow$ output the vertices of a **maximal matching**.

- This algorithm is a 2-approximation for MINIMUM VERTEX COVER.

Preliminaries: approximation algorithms

- Given a (typically NP-hard) *minimization* problem Π, we say that ALG is an α-approximation algorithm for Π (with $\alpha \geq 1$) if for any instance I of Π,

$$ALG(I) \leq \alpha \cdot OPT(I).$$

- **Example**:

  MINIMUM VERTEX COVER

  Input: An undirected graph $G = (V, E)$.

  Output: A subset $S \subseteq V$ such that for each $\{u, v\} \in E$, at least one of u and v is in S, and such that $|S|$ is minimized.

- Approximation algorithm for MINIMUM VERTEX COVER:
  $\longrightarrow$ output the vertices of a **maximal matching**.

- This algorithm is a 2-approximation for MINIMUM VERTEX COVER.

Preliminaries: approximation algorithms

- Given a (typically NP-hard) *minimization* problem Π, we say that ALG is an α-approximation algorithm for Π (with $\alpha \geq 1$) if for any instance I of Π,

$$ALG(I) \leq \alpha \cdot OPT(I).$$

- **Example**:

  MINIMUM VERTEX COVER

  Input: An undirected graph $G = (V, E)$.

  Output: A subset $S \subseteq V$ such that for each $\{u, v\} \in E$, at least one of u and v is in S, and such that $|S|$ is minimized.

- Approximation algorithm for MINIMUM VERTEX COVER:
  $\longrightarrow$ output the vertices of a **maximal matching.**

- This algorithm is a 2-approximation for MINIMUM VERTEX COVER.

Preliminaries: approximation algorithms

- Given a (typically NP-hard) *minimization* problem Π, we say that ALG is an α-approximation algorithm for Π (with $\alpha \geq 1$) if for any instance I of Π,

$$ALG(I) \leq \alpha \cdot OPT(I).$$

- **Example**:

  MINIMUM VERTEX COVER

  Input: An undirected graph $G = (V, E)$.

  Output: A subset $S \subseteq V$ such that for each $\{u, v\} \in E$, at least one of u and v is in S, and such that $|S|$ is minimized.

- Approximation algorithm for MINIMUM VERTEX COVER:
  $\longrightarrow$ output the vertices of a **maximal matching.**

- This algorithm is a 2-approximation for MINIMUM VERTEX COVER.

Preliminaries (II): hardness of approximation

- **Class APX (Approximable):**

  an NP-hard optimization problem is in APX if it can be approximated within a constant factor.

  Example: MINIMUM VERTEX COVER

- **Class PTAS (Polynomial-Time Approximation Scheme):**

  an NP-hard optimization problem is in PTAS if it can be approximated within a constant factor $1+\varepsilon$, for all $\varepsilon > 0$ (the best one can hope for an NP-hard problem).

  Example: MAXIMUM KNAPSACK

- We know that

$$\text{PTAS} \subsetneq \text{APX} \quad \text{(again, MIN VERTEX COVER!)}$$

- Thus, if Π is an optimization problem:

$$\Pi \text{ is APX-hard} \quad \Rightarrow \quad \Pi \notin \text{PTAS}$$

Preliminaries (II): hardness of approximation

- **Class APX (Approximable):**

  an NP-hard optimization problem is in APX if it can be approximated within a constant factor.

  Example: MINIMUM VERTEX COVER

- **Class PTAS (Polynomial-Time Approximation Scheme):**

  an NP-hard optimization problem is in PTAS if it can be approximated within a constant factor $1 + \varepsilon$, for all $\varepsilon > 0$ (the best one can hope for an NP-hard problem).

  Example: MAXIMUM KNAPSACK

- We know that

$$\text{PTAS} \subsetneq \text{APX} \quad \text{(again, MIN VERTEX COVER!)}$$

- Thus, if Π is an optimization problem:

$$\Pi \text{ is APX-hard} \Rightarrow \Pi \notin \text{PTAS}$$

Preliminaries (II): hardness of approximation

- **Class APX (Approximable):**

  an NP-hard optimization problem is in APX if it can be approximated within a constant factor.

  Example: MINIMUM VERTEX COVER

- **Class PTAS (Polynomial-Time Approximation Scheme):**

  an NP-hard optimization problem is in PTAS if it can be approximated within a constant factor $1 + \varepsilon$, for all $\varepsilon > 0$ (the best one can hope for an NP-hard problem).

  Example: MAXIMUM KNAPSACK

- We know that

  $$\text{PTAS} \subsetneq \text{APX} \quad \text{(again, MIN VERTEX COVER!)}$$

- Thus, if Π is an optimization problem:

  $$\Pi \text{ is APX-hard} \;\Rightarrow\; \Pi \notin \text{PTAS}$$

1- MAXIMUM *d*-DEGREE-BOUNDED CONNECTED SUBGRAPH

Definition of the problem

- **MAXIMUM d-DEGREE-BOUNDED CONNECTED SUBGRAPH ($\mathbf{MDBCS}_d$):**

  Input:
  - an undirected graph $G = (V, E)$,
  - an integer $d \geq 2$, and
  - a weight function $\omega : E \to \mathbb{R}^+$.

  Output:
  a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. $H = G[E']$
  - is **connected**, and
  - satisfies $\Delta(H) \leq d$.

- It is one of the classical **NP**-hard problems of [Garey and Johnson, Computers and Intractability, 1979].
- If the output subgraph is not required to be connected, the problem is in **P** for any d (using matching techniques).
- For *fixed* $d = 2$ it is the well known LONGEST PATH (OR CYCLE) problem.

Definition of the problem

- **MAXIMUM d-DEGREE-BOUNDED CONNECTED SUBGRAPH ($\mathbf{MDBCS}_d$)**:

  Input:
  - an undirected graph $G = (V, E)$,
  - an integer $d \geq 2$, and
  - a weight function $\omega : E \to \mathbb{R}^+$.

  Output:
  a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. $H = G[E']$
  - is **connected**, and
  - satisfies $\Delta(H) \leq d$.

- It is one of the classical **NP**-hard problems of [Garey and Johnson, Computers and Intractability, 1979].
- If the output subgraph is not required to be connected, the problem is in **P** for any d (using matching techniques).
- For *fixed* $d = 2$ it is the well known LONGEST PATH (OR CYCLE) problem.

Definition of the problem

- **MAXIMUM *d*-DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS_d):**

  Input:
  - an undirected graph $G = (V, E)$,
  - an integer $d \geq 2$, and
  - a weight function $\omega : E \rightarrow \mathbb{R}^+$.

  Output:
  a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. $H = G[E']$
  - is **connected**, and
  - satisfies $\Delta(H) \leq d$.

- It is one of the classical **NP**-hard problems of [Garey and Johnson, Computers and Intractability, 1979].
- If the output subgraph is not required to be connected, the problem is in **P** for any *d* (using matching techniques).
- For *fixed* $d = 2$ it is the well known LONGEST PATH (OR CYCLE) problem.

Definition of the problem

- **MAXIMUM d-DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS$_d$):**

  Input:
  - an undirected graph $G = (V, E)$,
  - an integer $d \geq 2$, and
  - a weight function $\omega : E \to \mathbb{R}^+$.

  Output:
  a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. $H = G[E']$
  - is **connected**, and
  - satisfies $\Delta(H) \leq d$.

- It is one of the classical **NP**-hard problems of [Garey and Johnson, Computers and Intractability, 1979].
- If the output subgraph is not required to be connected, the problem is in **P** for any d (using matching techniques).
- For *fixed* $d = 2$ it is the well known LONGEST PATH (OR CYCLE) problem.

Definition of the problem

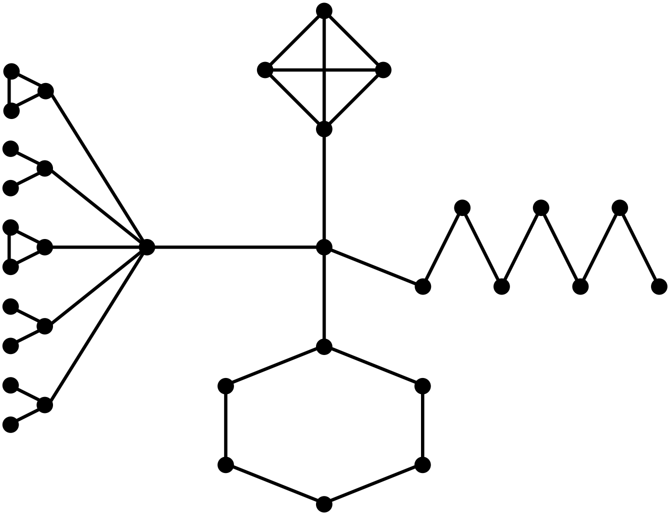
- **MAXIMUM *d*-DEGREE-BOUNDED CONNECTED SUBGRAPH (MDBCS$_d$):**

  Input:
  - an undirected graph $G = (V, E)$,
  - an integer $d \geq 2$, and
  - a weight function $\omega : E \rightarrow \mathbb{R}^+$.

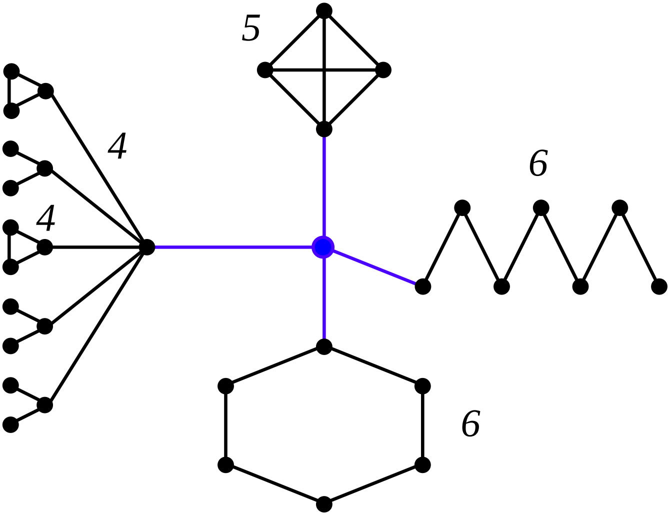
  Output:
  a subset of edges $E' \subseteq E$ of **maximum weight**, s.t. $H = G[E']$
  - is **connected**, and
  - satisfies $\Delta(H) \leq d$.

- It is one of the classical **NP**-hard problems of [Garey and Johnson, Computers and Intractability, 1979].
- If the output subgraph is not required to be connected, the problem is in **P** for any d (using matching techniques).
- For *fixed* $d = 2$ it is the well known **LONGEST PATH (OR CYCLE)** problem.

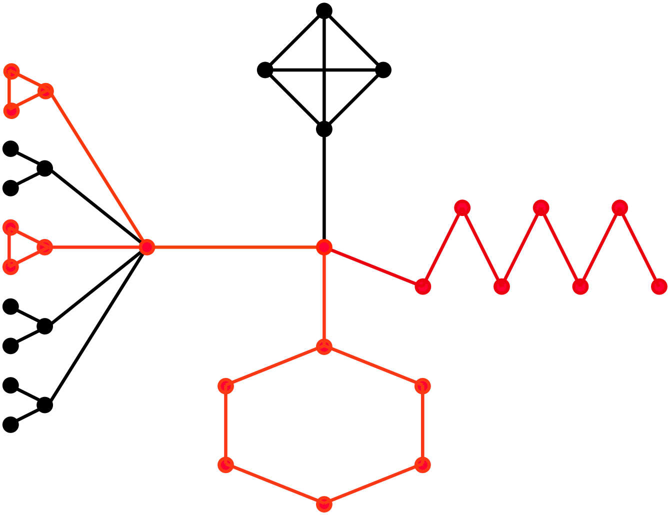
Example with $d = 3$, $\omega(e) = 1$ for all $e \in E(G)$



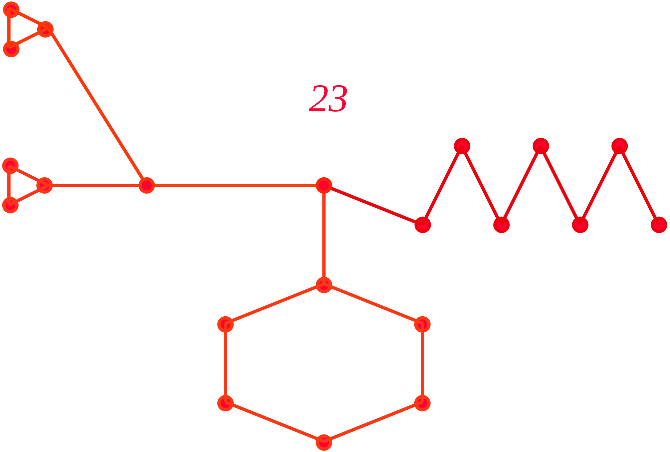
Example with $d = 3$ (II)



Example with $d = 3$ (III)



Example with $d = 3$ (IV)



State of the art

To the best of our knowledge, there were no results in the literature except for the case $d = 2$, a.k.a. the **LONGEST PATH** problem:

- **Approximation algorithms**:
  $\mathcal{O}\left(\frac{n}{\log n}\right)$-approximation, using the **color-coding** method.
  [N. Alon, R. Yuster and U. Zwick, STOC'94].
  $\mathcal{O}\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$-approximation.
  [A. Björklund and T. Husfeldt, SIAM J. Computing'03].

- **Hardness results**:
  It does not accept *any* constant-factor approximation.
  [D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97].

State of the art

To the best of our knowledge, there were no results in the literature except for the case $d = 2$, a.k.a. the **LONGEST PATH** problem:

- **Approximation algorithms**:
  $\mathcal{O}\left(\frac{n}{\log n}\right)$-approximation, using the **color-coding** method.
  [N. Alon, R. Yuster and U. Zwick, STOC'94].
  $\mathcal{O}\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$-approximation.
  [A. Björklund and T. Husfeldt, SIAM J. Computing'03].

- **Hardness results**:
  It does not accept *any* constant-factor approximation.
  [D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97].

State of the art

To the best of our knowledge, there were no results in the literature except for the case $d = 2$, a.k.a. the **LONGEST PATH** problem:

- **Approximation algorithms**:
  $\mathcal{O}\left(\frac{n}{\log n}\right)$-approximation, using the **color-coding** method.
  [N. Alon, R. Yuster and U. Zwick, STOC'94].
  $\mathcal{O}\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$-approximation.
  [A. Björklund and T. Husfeldt, SIAM J. Computing'03].

- **Hardness results**:
  It does not accept *any* constant-factor approximation.
  [D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97].

State of the art

To the best of our knowledge, there were no results in the literature except for the case $d = 2$, a.k.a. the **LONGEST PATH** problem:

- **Approximation algorithms**:
  $\mathcal{O}\left(\frac{n}{\log n}\right)$-approximation, using the **color-coding** method.
  [N. Alon, R. Yuster and U. Zwick, STOC'94].
  $\mathcal{O}\left(n\left(\frac{\log\log n}{\log n}\right)^2\right)$-approximation.
  [A. Björklund and T. Husfeldt, SIAM J. Computing'03].

- **Hardness results**:
  It does not accept *any* constant-factor approximation.
  [D. Karger, R. Motwani and G. Ramkumar, Algorithmica'97].

Our results

- **Approximation algorithms** ($n = |V(G)|$, $m = |E(G)|$):
  - $\min\{\frac{n}{2}, \frac{m}{d}\}$-approximation algorithm for **weighted** graphs.
  - $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$-approximation algorithm for **unweighted** graphs, using *color coding*.
  - when G **accepts a low-degree spanning tree**, in terms of d, then MDBCS$_d$ can be approximated within a **small constant factor**.
- **Hardness results**:
  - For each fixed $d \geq 2$, MDBCS$_d$ does not accept *any* constant-factor approximation in general graphs.

Our results

- **Approximation algorithms** ($n = |V(G)|$, $m = |E(G)|$):
  - $\min\{\frac{n}{2}, \frac{m}{d}\}$-approximation algorithm for **weighted** graphs.
  - $\min\{\frac{m}{\log n}, \frac{nd}{2 \log n}\}$-approximation algorithm for **unweighted** graphs, using *color coding*.
  - when G **accepts a low-degree spanning tree**, in terms of d, then MDBCS_d can be approximated within a **small constant factor**.
- **Hardness results**:
  - For each fixed $d \geq 2$, MDBCS_d does not accept *any* constant-factor approximation in general graphs.

Our results

- **Approximation algorithms** ($n = |V(G)|$, $m = |E(G)|$):
  - $\min\{\frac{n}{2}, \frac{m}{d}\}$-approximation algorithm for **weighted** graphs.
  - $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$-approximation algorithm for **unweighted** graphs, using *color coding*.
  - when G **accepts a low-degree spanning tree**, in terms of d, then MDBCS_d can be approximated within a **small constant factor**.

- **Hardness results**:
  - For each fixed $d \geq 2$, MDBCS_d does not accept *any* constant-factor approximation in general graphs.

Our results

- **Approximation algorithms** ($n = |V(G)|$, $m = |E(G)|$):
  - $\min\{\frac{n}{2}, \frac{m}{d}\}$-approximation algorithm for **weighted** graphs.
  - $\min\{\frac{m}{\log n}, \frac{nd}{2\log n}\}$-approximation algorithm for **unweighted** graphs, using *color coding*.
  - when G **accepts a low-degree spanning tree**, in terms of d, then MDBCS_d can be approximated within a **small constant factor**.
- **Hardness results**:
  - For each fixed $d \geq 2$, MDBCS_d does not accept *any* constant-factor approximation in general graphs.

Our results

- **Approximation algorithms** ($n = |V(G)|$, $m = |E(G)|$):
  - $\min\{\frac{n}{2}, \frac{m}{d}\}$-approximation algorithm for **weighted** graphs.
  - $\min\{\frac{m}{\log n}, \frac{nd}{2 \log n}\}$-approximation algorithm for **unweighted** graphs, using *color coding*.
  - when G **accepts a low-degree spanning tree**, in terms of d, then MDBCS$_d$ can be approximated within a **small constant factor**.
- **Hardness results**:
  - For each fixed $d \geq 2$, MDBCS$_d$ does not accept *any* constant-factor approximation in general graphs.

Approximation algorithm for weighted graphs

Input: undirected graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \geq 2$. Let $n = |V|$, $m = |E|$.

F: set of d heaviest edges in G, with weight $\omega(F)$.
W: set of endpoints of those edges. Let $H = (W, F)$.

Description of the algorithm: Two cases according to $H = (W, F)$:

(1) If $H = (W, F)$ is connected, the algorithm returns H.
Claim: this yields a $\min\{n/2, m/d\}$-approximation.

Proof.
Suppose an optimal solution consists of m^* edges of total weight ω^*. Then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F can not be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d = \frac{\omega^*}{m/d}$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

(2) If $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components, we *glue* them in $k - 1$ phases.

Approximation algorithm for weighted graphs

Input: undirected graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \geq 2$. Let $n = |V|$, $m = |E|$.

F: set of d heaviest edges in G, with weight $\omega(F)$.
W: set of endpoints of those edges. Let $H = (W, F)$.

Description of the algorithm: Two cases according to $H = (W, F)$:

(1) If $H = (W, F)$ is connected, the algorithm returns H.
Claim: this yields a $\min\{n/2, m/d\}$-approximation.

Proof.
Suppose an optimal solution consists of m^* edges of total weight ω^*. Then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F can not be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d = \frac{\omega^*}{m/d}$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

(2) If $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components, we *glue* them in $k - 1$ phases.

Approximation algorithm for weighted graphs

Input: undirected graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \geq 2$. Let $n = |V|$, $m = |E|$.

F: set of d heaviest edges in G, with weight $\omega(F)$.
W: set of endpoints of those edges. Let $H = (W, F)$.

Description of the algorithm: Two cases according to $H = (W, F)$:

(1) If $H = (W, F)$ is connected, the algorithm returns H.
Claim: this yields a $\min\{n/2, m/d\}$-approximation.

Proof.
Suppose an optimal solution consists of m^* edges of total weight ω^*. Then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F can not be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d = \frac{\omega^*}{m/d}$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

(2) If $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components, we *glue* them in $k - 1$ phases.

Approximation algorithm for weighted graphs

Input: undirected graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \geq 2$. Let $n = |V|$, $m = |E|$.

F: set of d heaviest edges in G, with weight $\omega(F)$.
W: set of endpoints of those edges. Let $H = (W, F)$.

Description of the algorithm: Two cases according to $H = (W, F)$:

(1) If $H = (W, F)$ is connected, the algorithm returns H.
Claim: this yields a $\min\{n/2, m/d\}$-approximation.

Proof.
Suppose an optimal solution consists of m^* edges of total weight ω^*. Then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F can not be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d = \frac{\omega^*}{m/d}$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

(2) If $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components, we *glue* them in $k - 1$ phases.

Approximation algorithm for weighted graphs

Input: undirected graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \geq 2$. Let $n = |V|$, $m = |E|$.

F: set of d heaviest edges in G, with weight $\omega(F)$.
W: set of endpoints of those edges. Let $H = (W, F)$.

Description of the algorithm: Two cases according to $H = (W, F)$:

(1) If $H = (W, F)$ is connected, the algorithm returns H.
Claim: this yields a $\min\{n/2, m/d\}$-approximation.

Proof.
Suppose an optimal solution consists of m^* edges of total weight ω^*. Then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F can not be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d = \frac{\omega^*}{m/d}$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

(2) If $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components, we *glue* them in $k - 1$ phases.

Approximation algorithm for weighted graphs

Input: undirected graph $G = (V, E)$, a weight function $\omega : E \to \mathbb{R}^+$, and an integer $d \geq 2$. Let $n = |V|$, $m = |E|$.

F: set of d heaviest edges in G, with weight $\omega(F)$.
W: set of endpoints of those edges. Let $H = (W, F)$.

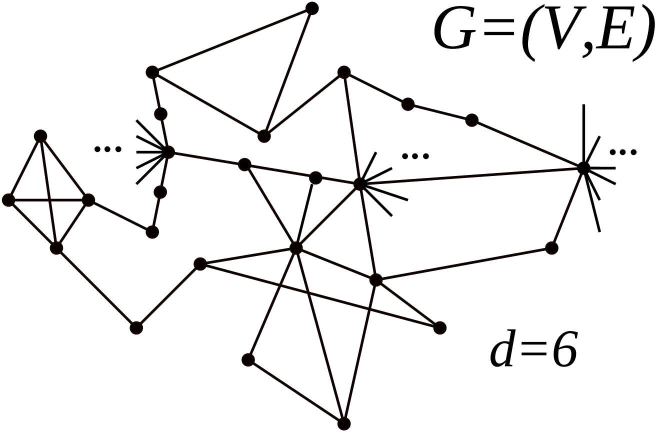
Description of the algorithm: Two cases according to $H = (W, F)$:

(1) If $H = (W, F)$ is connected, the algorithm returns H.
Claim: this yields a $\min\{n/2, m/d\}$-approximation.

Proof.
Suppose an optimal solution consists of m^* edges of total weight ω^*. Then $ALG = \omega(F) \geq \frac{\omega^*}{m^*} \cdot d$, since by the choice of F the average weight of the edges in F can not be smaller than the average weight of the edges of an optimal solution. As $m^* \leq m$ and $m^* \leq dn/2$, we get that $ALG \geq \frac{\omega^*}{m} \cdot d = \frac{\omega^*}{m/d}$ and $ALG \geq \frac{\omega^*}{dn/2} \cdot d = \frac{\omega^*}{n/2}$.

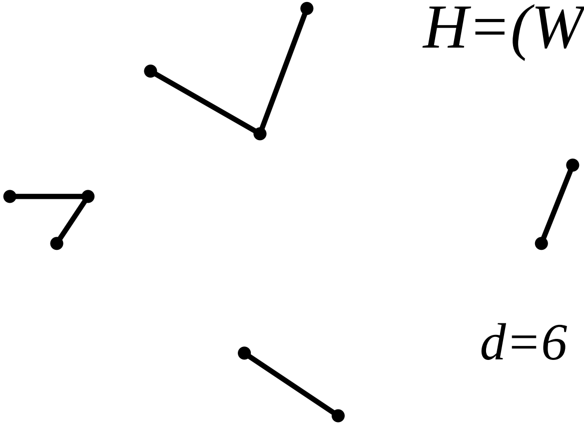
(2) If $H = (W, F)$ consists of a collection $\mathcal{F}$ of k connected components, we *glue* them in $k - 1$ phases.

Example of the algorithm for weighted graphs



- Given a weighted graph $G = (V, E)$ and an integer d...

Example of the algorithm for weighted graphs

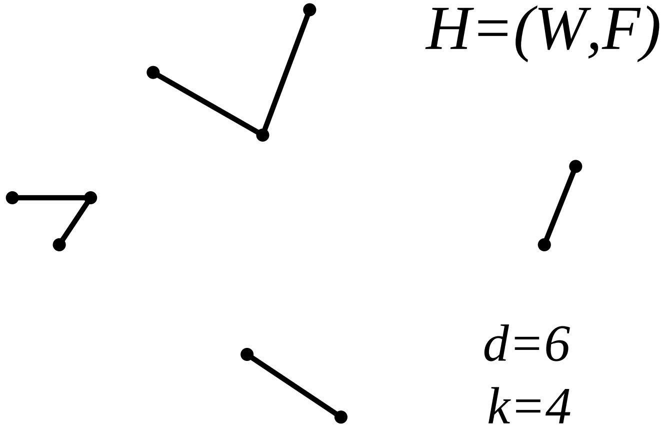


$H=(W,F)$

$d=6$

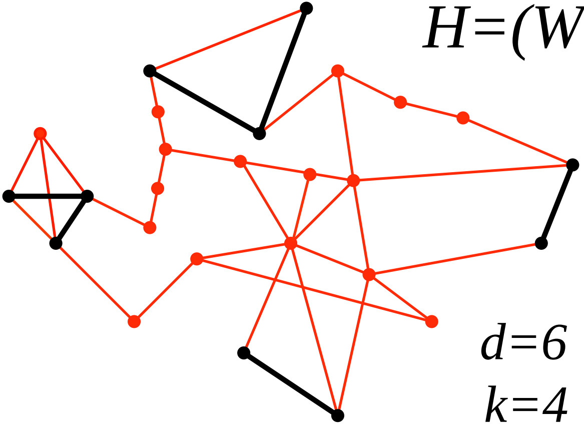
- Let $H = (W, F)$ be the graph induced by the d heaviest edges.

Example of the algorithm for weighted graphs



- Assume H has $k > 1$ connected components.

Example of the algorithm for weighted graphs



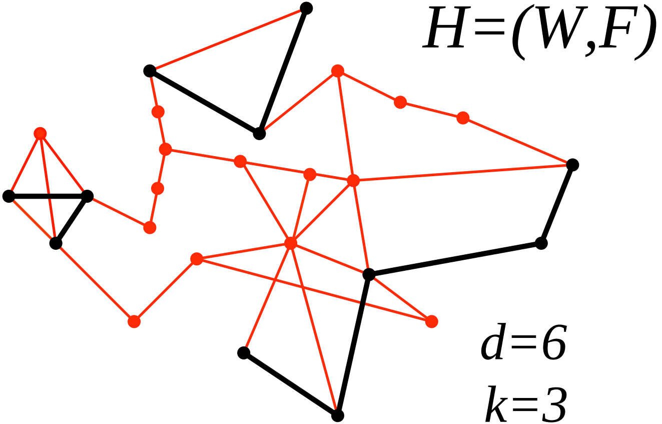
$H=(W,F)$

$d=6$

$k=4$

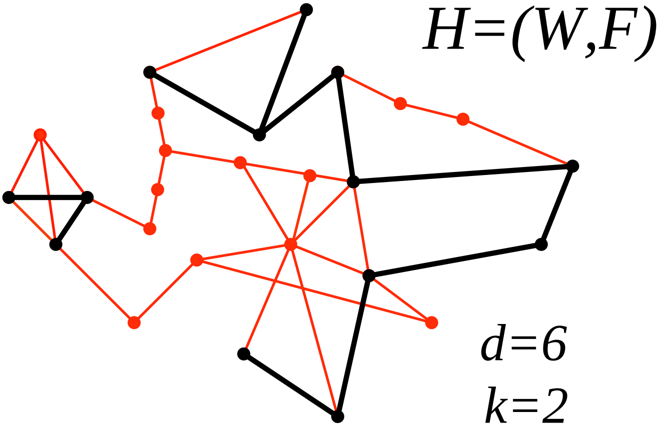
- We compute the distance in G between each pair of components.

Example of the algorithm for weighted graphs



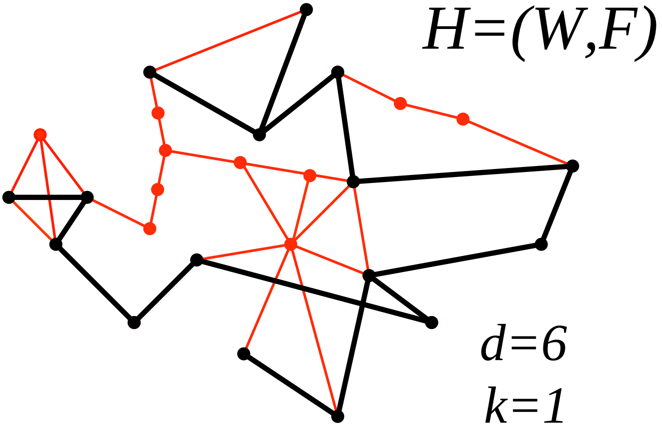
- We add to H a path between a pair of closest vertices.

Example of the algorithm for weighted graphs



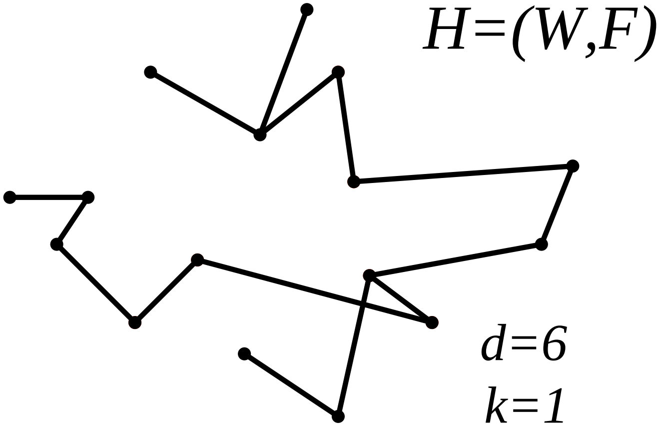
- We repeat these two steps inductively...

Example of the algorithm for weighted graphs



- Until the graph H is connected.

Example of the algorithm for weighted graphs



- The algorithm outputs this graph H.

Analysis of the algorithm

(a) Running time: clearly polynomial.

(b) Correctness:
- The output subgraph is connected.
- **Claim**: after i phases, $\Delta(H) \leq d - k + i + 1$.
  The proof is done by induction. When $i = k - 1$ we get $\Delta(H) \leq d$.

(c) Approximation ratio: follows from case (1).

Analysis of the algorithm

(a) **Running time**: clearly polynomial.

(b) **Correctness**:

- The output subgraph is connected.
- **Claim**: after i phases, $\Delta(H) \leq d - k + i + 1$.
  The proof is done by induction. When $i = k - 1$ we get $\Delta(H) \leq d$.

(c) **Approximation ratio**: follows from case (1).

Analysis of the algorithm

(a) Running time: clearly polynomial.

(b) Correctness:

- The output subgraph is connected.
- **Claim**: after i phases, $\Delta(H) \leq d - k + i + 1$.
  The proof is done by induction. When $i = k - 1$ we get $\Delta(H) \leq d$.

(c) Approximation ratio: follows from case (1).

Analysis of the algorithm

(a) Running time: clearly polynomial.

(b) Correctness:

- The output subgraph is connected.
- **Claim**: after i phases, $\Delta(H) \leq d - k + i + 1$.
  The proof is done by induction. When $i = k - 1$ we get $\Delta(H) \leq d$.

(c) Approximation ratio: follows from case (1).

2- Minimum Subgraph of Minimum Degree $\geq d$

Definition of the problem

- **MINIMUM SUBGRAPH OF MINIMUM DEGREE $\geq d$ (MSMD$_d$)**:

  Input: an undirected graph $G = (V, E)$ and an integer $d \geq 3$.

  Output: a subset $S \subseteq V$ with $\delta(G[S]) \geq d$, s.t. $|S|$ is minimum.

- For $d = 2$ it is the GIRTH problem (find the length of a shortest cycle), which is in P.

- Motivation: close relation with DENSE k-SUBGRAPH problem and TRAFFIC GROOMING problem in optical networks.

Definition of the problem

- **MINIMUM SUBGRAPH OF MINIMUM DEGREE $\geq d$ (MSMD$_d$)**:

  Input: an undirected graph $G = (V, E)$ and an integer $d \geq 3$.

  Output: a subset $S \subseteq V$ with $\delta(G[S]) \geq d$, s.t. $|S|$ is minimum.

- For $d = 2$ it is the GIRTH problem (find the length of a shortest cycle), which is in P.

- Motivation: close relation with DENSE k-SUBGRAPH problem and TRAFFIC GROOMING problem in optical networks.

Definition of the problem

- **Minimum Subgraph of Minimum Degree** $\geq d$ **(MSMD$_d$)**:

  Input: an undirected graph $G = (V, E)$ and an integer $d \geq 3$.

  Output: a subset $S \subseteq V$ with $\delta(G[S]) \geq d$, s.t. $|S|$ is minimum.

- For $d = 2$ it is the Girth problem (find the length of a shortest cycle), which is in P.

- Motivation: close relation with Dense k-Subgraph problem and Traffic Grooming problem in optical networks.

Definition of the problem

- **MINIMUM SUBGRAPH OF MINIMUM DEGREE** $\geq d$ **(MSMD$_d$)**:

  Input: an undirected graph $G = (V, E)$ and an integer $d \geq 3$.

  Output: a subset $S \subseteq V$ with $\delta(G[S]) \geq d$, s.t. $|S|$ is minimum.

- For $d = 2$ it is the GIRTH problem (find the length of a shortest cycle), which is in P.

- Motivation: close relation with DENSE k-SUBGRAPH problem and TRAFFIC GROOMING problem in optical networks.

State of the art + our results

- This problem was first introduced in [O. Amini, I. S. and S. Saurabh, IWPEC'08].
  - W[1]-hard in general graphs, for $d \geq 3$.
  - FPT in minor-closed classes of graphs.
- Our results:
  - MSMD_d is not in APX for any $d \geq 3$.
  - $\mathcal{O}(n/\log n)$-approximation algorithm for minor-closed classes of graphs, using a structural result and dynamic programming.

State of the art + our results

- This problem was first introduced in
  [O. Amini, I. S. and S. Saurabh, IWPEC'08].
  - W[1]-hard in general graphs, for $d \geq 3$.
  - FPT in minor-closed classes of graphs.
- Our results:
  - MSMD_d is not in APX for any $d \geq 3$.
  - $\mathcal{O}(n/\log n)$-approximation algorithm for minor-closed classes of graphs, using a structural result and dynamic programming.

State of the art + our results

- This problem was first introduced in [O. Amini, I. S. and S. Saurabh, IWPEC'08].
  - W[1]-hard in general graphs, for $d \geq 3$.
  - FPT in minor-closed classes of graphs.
- Our results:
  - MSMD_d is not in APX for any $d \geq 3$.
  - $\mathcal{O}(n/\log n)$-approximation algorithm for minor-closed classes of graphs, using a structural result and dynamic programming.

Hardness result

Idea of the proof for $d = 3$

(1) First we will see that $\mathrm{MSMD}_3 \notin \mathrm{PTAS}$.

(2) Then we will see that $\mathrm{MSMD}_3 \notin \mathrm{APX}$.

(1) *MSMD*$_3$ is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance H of VERTEX COVER $\rightarrow$ Instance G of MSMD$_3$

- We will see that

  $$\text{PTAS for MSMD}_3 \Rightarrow \text{PTAS for VERTEX COVER}$$

- And so,

  $$\nexists \text{ PTAS for MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$ and $\delta(H) \geq 3$.

(1) *MSMD*$_3$ is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance H of VERTEX COVER $\rightarrow$ Instance G of MSMD$_3$

- We will see that

  PTAS for MSMD$_3$ $\Rightarrow$ PTAS for VERTEX COVER

- And so,

  $\nexists$ PTAS for MSMD$_3$

- We can suppose $|E(H)| = 3 \cdot 2^m$ and $\delta(H) \geq 3$.

(1) *MSMD*$_3$ is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance H **of VERTEX COVER** $\rightarrow$ **Instance** G **of MSMD**$_3$

- We will see that

  $$\text{PTAS for MSMD}_3 \Rightarrow \text{PTAS for VERTEX COVER}$$

- And so,

  $$\nexists\ \text{PTAS for MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$ and $\delta(H) \geq 3$.

(1) *MSMD*$_3$ is not in *PTAS*

- Reduction from VERTEX COVER:

  Instance *H* of VERTEX COVER $\rightarrow$ **Instance *G* of MSMD$_3$**

- We will see that

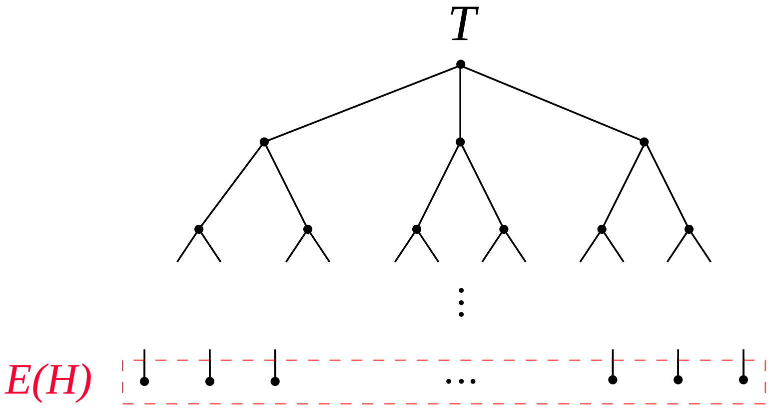
  $$\text{PTAS for MSMD}_3 \Rightarrow \text{PTAS for VERTEX COVER}$$

- And so,

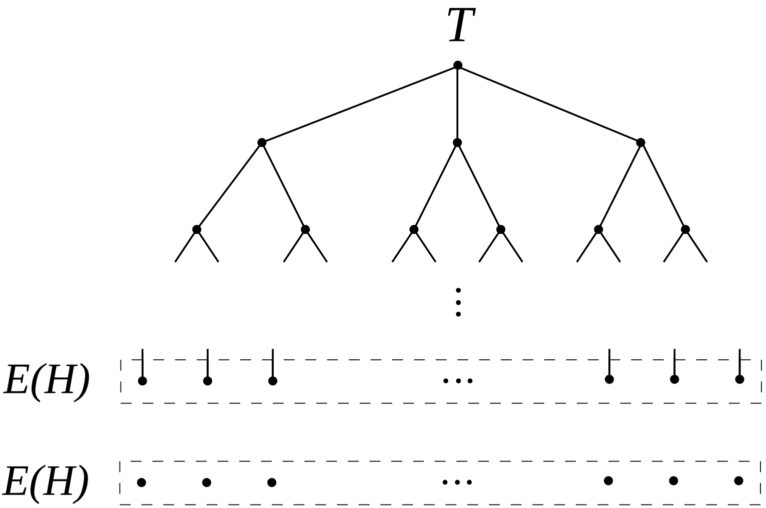
  $$\nexists\ \text{PTAS for MSMD}_3$$

- We can suppose $|E(H)| = 3 \cdot 2^m$ and $\delta(H) \geq 3$.

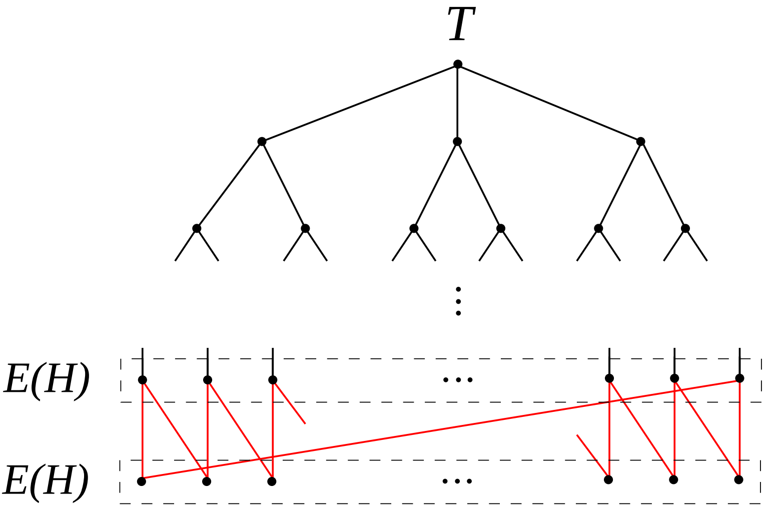
We build a complete ternary tree with $|E(H)| = 3 \cdot 2^m$ leaves:



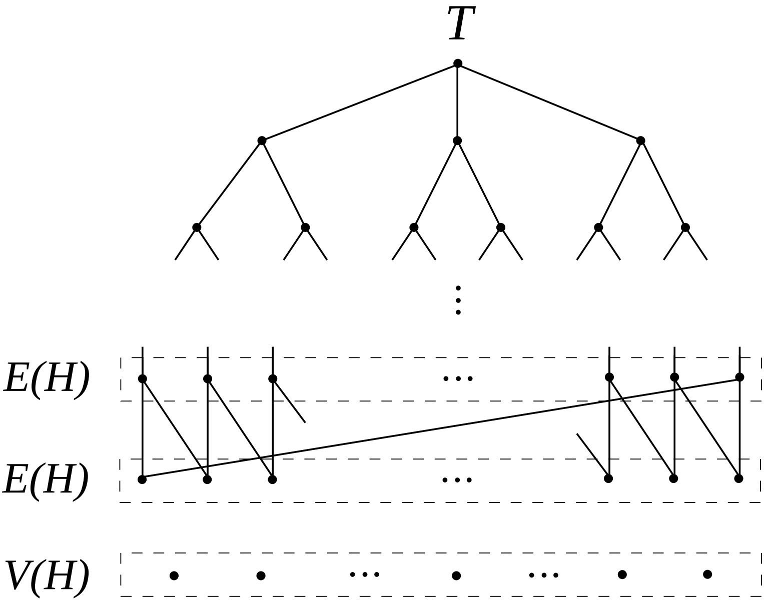
We add a copy of the set of leaves $E(H)$:



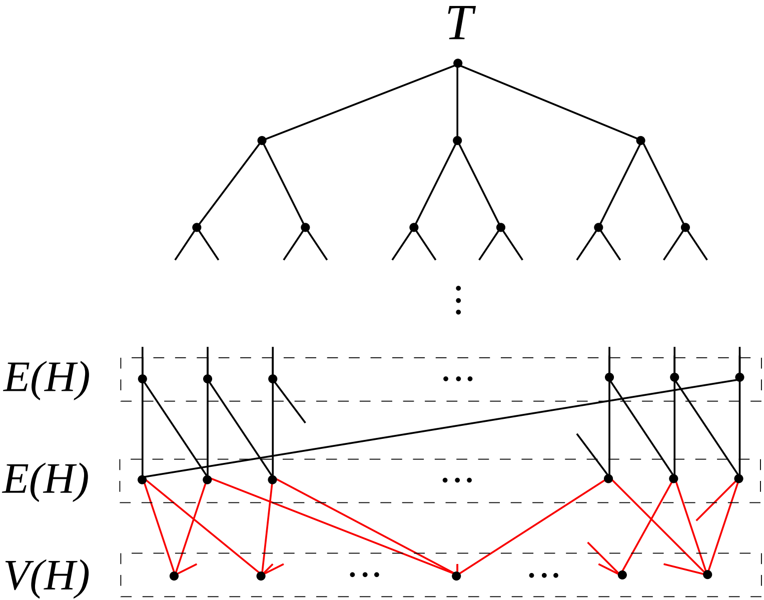
We join both sets with a Hamiltonian cycle (for technical reasons):



We add all the vertices of H:



We add the incidence relations between $E(H)$ and $V(H) \rightarrow G$:



(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$

- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!

- Thus,

$$OPT_{\text{MSMD}_3}(G) = OPT_{\text{VC}}(H) + |V(G \setminus V(H))| =$$
$$= OPT_{\text{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for MSMD}_3 \;\Rightarrow\; \text{PTAS for VERTEX COVER}$$

(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$
- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  → this is **exactly** VERTEX COVER in H !!
- Thus,

$$OPT_{\text{MSMD}_3}(G) = OPT_{\text{VC}}(H) + |V(G \setminus V(H))| =$$
$$= OPT_{\text{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for MSMD}_3 \quad \Rightarrow \quad \text{PTAS for VERTEX COVER}$$

(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$
- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!

- Thus,

$$OPT_{\text{MSMD}_3}(G) = OPT_{\text{VC}}(H) + |V(G \setminus V(H))| =$$
$$= OPT_{\text{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for MSMD}_3 \quad \Rightarrow \quad \text{PTAS for VERTEX COVER}$$

(1) MSMD_3 is not in PTAS

- If we touch a vertex of $G \setminus V(H)$, we have to touch all the vertices of $G \setminus V(H)$
- Thus, MSMD_3 in G is equivalent to minimize the number of selected vertices in $V(H)$

  $\rightarrow$ this is **exactly** VERTEX COVER in H !!
- Thus,

$$OPT_{\text{MSMD}_3}(G) = OPT_{\text{VC}}(H) + |V(G \setminus V(H))| =$$
$$= OPT_{\text{VC}}(H) + 9 \cdot 2^m$$

- This clearly proves that

$$\text{PTAS for MSMD}_3 \;\Rightarrow\; \text{PTAS for VERTEX COVER}$$

(2) $MSMD_3$ is not in APX

- Let $\alpha > 1$ be the factor of inapproximability of $MSMD_3$
- We use a technique called **error amplification**:
  - We build a sequence of families of graphs $\mathcal{G}_k$, such that $MSMD_3$ is hard to approximate in $\mathcal{G}_k$ within a factor α^k
  - This proves that the problem is not in APX

    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)
- Let $G_1 = G$.

  We explain the construction of G_2: first take our graph G and...

(2) MSMD_3 is not in APX

- Let $\alpha > 1$ be the factor of inapproximability of MSMD_3
- We use a technique called **error amplification**:
  - We build a sequence of families of graphs $\mathcal{G}_k$, such that MSMD_3 is hard to approximate in $\mathcal{G}_k$ within a factor α^k
  - This proves that the problem is not in APX

    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)

- Let $G_1 = G$.
  We explain the construction of G_2: first take our graph G and...

(2) $MSMD_3$ is not in APX

- Let $\alpha > 1$ be the factor of inapproximability of $MSMD_3$
- We use a technique called **error amplification**:
  - We build a sequence of families of graphs $\mathcal{G}_k$, such that $MSMD_3$ is hard to approximate in $\mathcal{G}_k$ within a factor α^k
  - This proves that the problem is not in APX

    (for any constant C, $\exists$ $k > 0$ such that $\alpha^k > C$)

- Let $G_1 = G$.
  We explain the construction of G_2: first take our graph G and...

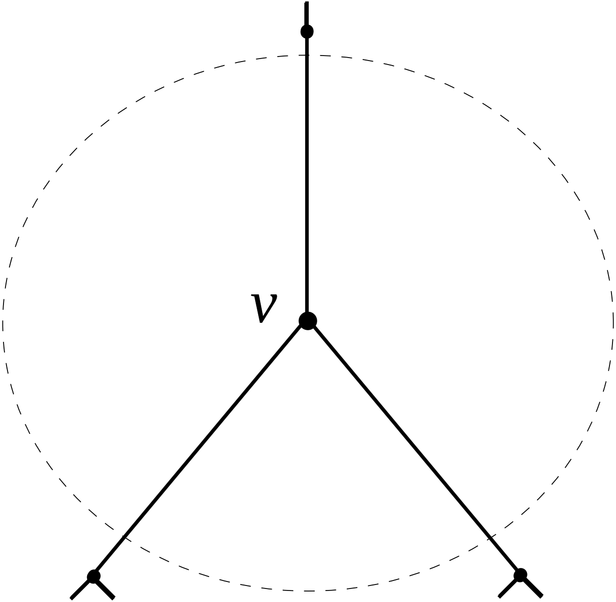
(2) MSMD_3 is not in APX

- Let $\alpha > 1$ be the factor of inapproximability of MSMD_3
- We use a technique called **error amplification**:
  - We build a sequence of families of graphs $\mathcal{G}_k$, such that MSMD_3 is hard to approximate in $\mathcal{G}_k$ within a factor α^k
  - This proves that the problem is not in APX

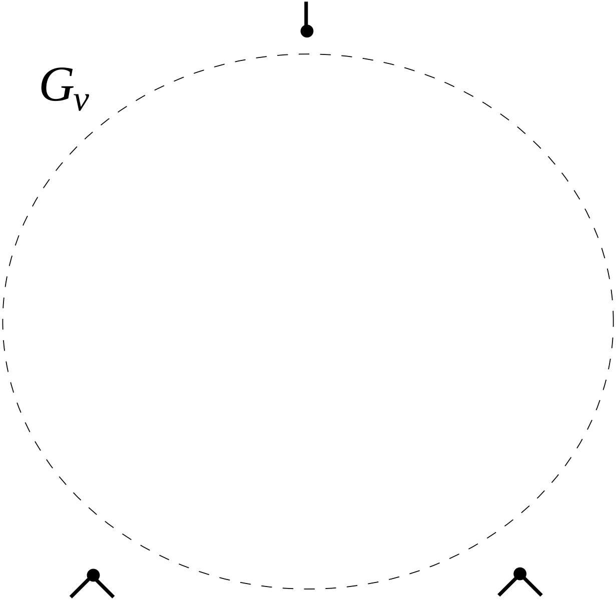
    (for any constant C, $\exists\ k > 0$ such that $\alpha^k > C$)

- Let $G_1 = G$.
  We explain the construction of G_2: first take our graph G and...

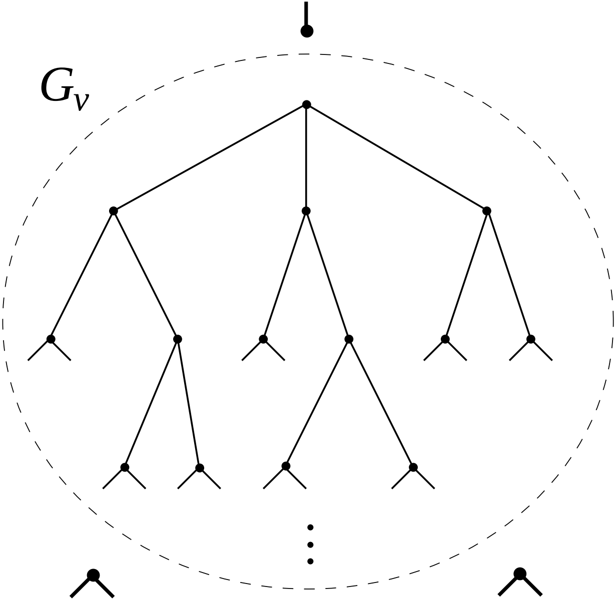
For any vertex v (note its degree by d_v):



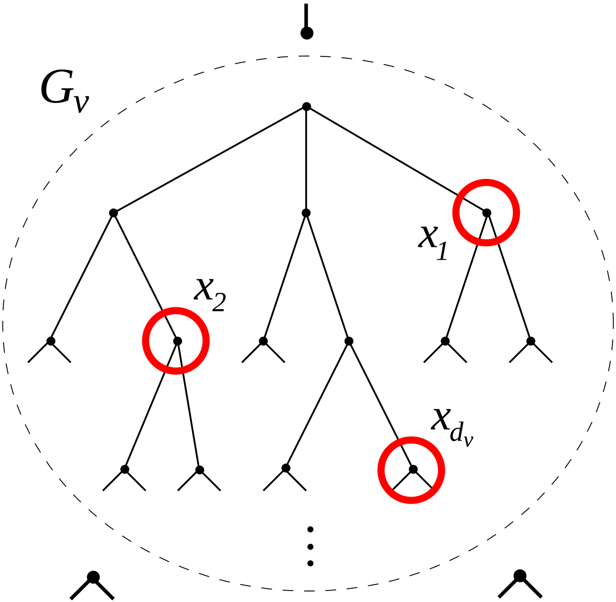
We will replace the vertex v with a graph G_v, built as follows:



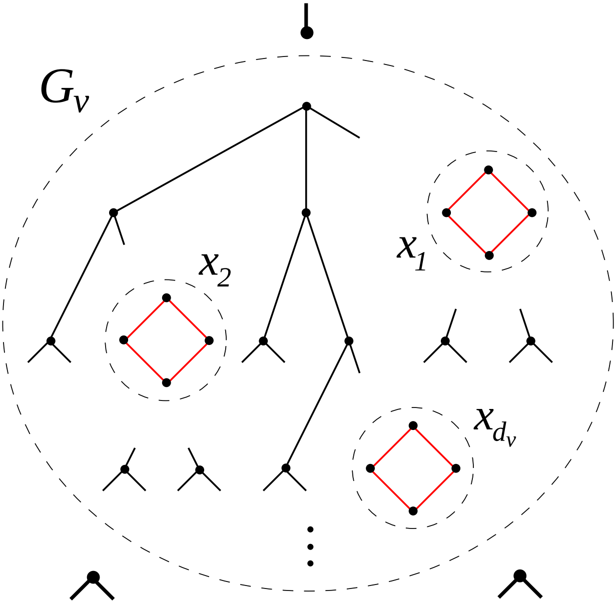
We begin by placing a copy of *G* (described before):



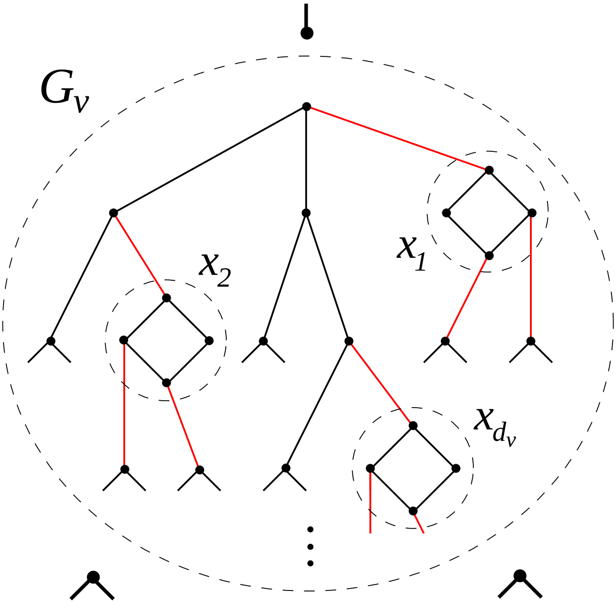
We select d_v vertices of degree 3 in $T \subset G$:



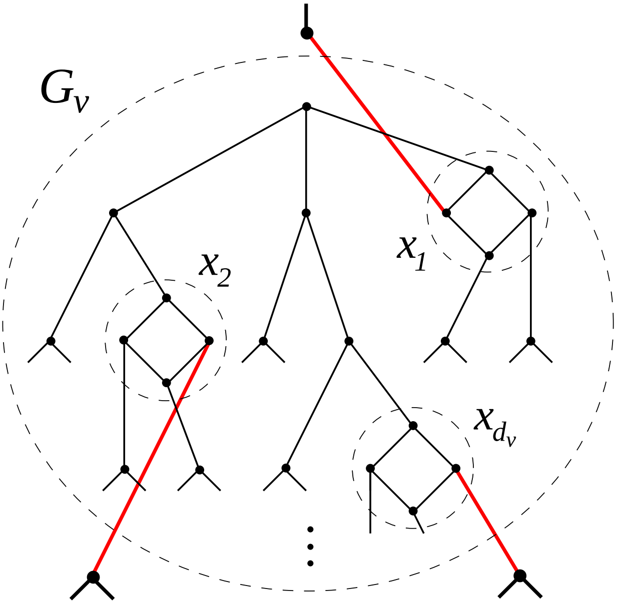
We replace each of these vertices x_i with a C_4:



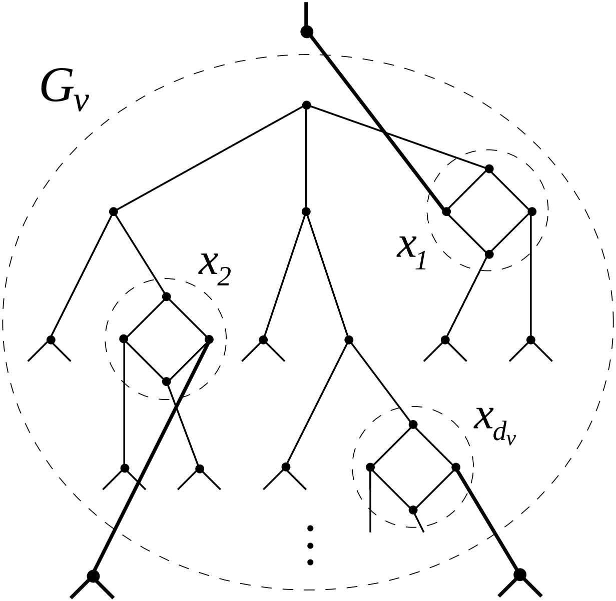
In each C_4, we join 3 of the vertices to the neighbors of x_i:



We join the d_v vertices of degree 2 to the d_v neighbors of v:



This construction for all $v \in G$ defines G_2:



(2) $MSMD_3$ is not in APX

- Once a vertex in one G_v is chosen $\rightarrow$ $MSMD_3$ in G_v (which is hard up to a constant α)
- But minimize the number of v's for which we touch G_v $\rightarrow$ $MSMD_3$ in G (which is also hard up to a constant α)
- Thus, in G_2 the problem is hard to approximate up to a factor $\alpha \cdot \alpha = \alpha^2$
- Inductively we prove that in G_k the problem is hard to approximate up to a factor α^k

(2) $MSMD_3$ is not in APX

- Once a vertex in one G_v is chosen $\rightarrow$ $MSMD_3$ in G_v

  (which is hard up to a constant α)

- But minimize the number of v's for which we touch G_v $\rightarrow$ $MSMD_3$ in G (which is also hard up to a constant α)

- Thus, in G_2 the problem is hard to approximate up to a factor $\alpha \cdot \alpha = \alpha^2$

- Inductively we prove that in G_k the problem is hard to approximate up to a factor α^k

(2) $MSMD_3$ is not in APX

- Once a vertex in one G_v is chosen $\rightarrow$ $MSMD_3$ in G_v

  (which is hard up to a constant α)

- But minimize the number of v's for which we touch G_v $\rightarrow$ $MSMD_3$ in G (which is also hard up to a constant α)

- Thus, in G_2 the problem is hard to approximate up to a factor $\alpha \cdot \alpha = \alpha^2$

- Inductively we prove that in G_k the problem is hard to approximate up to a factor α^k

Approximation algorithm for minor free graphs

Recall: graph minors

- H is a contraction of G ($H \preceq_c G$) if H occurs from G after applying a series of edge contractions.
- H is a minor of G ($H \preceq_m G$) if H is the contraction of some subgraph of G.
- A graph class $\mathcal{G}$ is minor closed if every minor of a graph in $\mathcal{G}$ is again in $\mathcal{G}$.
- A graph class $\mathcal{G}$ is H-minor-free (or, excludes H as a minor) if no graph in $\mathcal{G}$ contains H as a minor.

Recall: graph minors

- H is a contraction of G ($H \preceq_c G$) if H occurs from G after applying a series of edge contractions.
- H is a minor of G ($H \preceq_m G$) if H is the contraction of some subgraph of G.
- A graph class $\mathcal{G}$ is minor closed if every minor of a graph in $\mathcal{G}$ is again in $\mathcal{G}$.
- A graph class $\mathcal{G}$ is H-minor-free (or, excludes H as a minor) if no graph in $\mathcal{G}$ contains H as a minor.

Recall: graph minors

- H is a contraction of G ($H \preceq_c G$) if H occurs from G after applying a series of edge contractions.
- H is a minor of G ($H \preceq_m G$) if H is the contraction of some subgraph of G.
- A graph class $\mathcal{G}$ is minor closed if every minor of a graph in $\mathcal{G}$ is again in $\mathcal{G}$.
- A graph class $\mathcal{G}$ is H-minor-free (or, excludes H as a minor) if no graph in $\mathcal{G}$ contains H as a minor.

Recall: graph minors

- H is a contraction of G ($H \preceq_c G$) if H occurs from G after applying a series of edge contractions.

- H is a minor of G ($H \preceq_m G$) if H is the contraction of some subgraph of G.

- A graph class $\mathcal{G}$ is minor closed if every minor of a graph in $\mathcal{G}$ is again in $\mathcal{G}$.

- A graph class $\mathcal{G}$ is H-minor-free (or, excludes H as a minor) if no graph in $\mathcal{G}$ contains H as a minor.

The problem is in P for graphs of *small* treewidth

Lemma

Let G be a graph on n vertices with treewidth at most t, and let d be a positive integer. Then in time $\mathcal{O}((d+1)^t(t+1)^{d^2}n)$ we can either

- *find a smallest subgraph of minimum degree at least d in G, or*
- *conclude that no such subgraph exists.*

Corollary

Let G be an n-vertex graph with treewidth $\mathcal{O}(\log n)$, and let d be a positive integer. Then in polynomial time one can either

- *find a smallest subgraph of minimum degree at least d in G, or*
- *conclude that no such subgraph exists.*

The problem is in P for graphs of *small* treewidth

Lemma

Let G be a graph on n vertices with treewidth at most t, and let d be a positive integer. Then in time $\mathcal{O}((d+1)^t (t+1)^{d^2} n)$ we can either

- *find a smallest subgraph of minimum degree at least d in G, or*
- *conclude that no such subgraph exists.*

Corollary

Let G be an n-vertex graph with treewidth $\mathcal{O}(\log n)$, and let d be a positive integer. Then in polynomial time one can either

- *find a smallest subgraph of minimum degree at least d in G, or*
- *conclude that no such subgraph exists.*

Nice partition of M-minor-free graphs

Theorem

For a fixed graph M, there is a constant c_M such that for any integer $k \geq 1$ and for every M-minor-free graph G, the vertices of G can be partitioned into $k+1$ sets such that any k of the sets induce a graph of treewidth at most $c_M k$.
Furthermore, such a partition can be found in polynomial time.

[E. Demaine, M.T. Hajiaghayi and K.C. Kawarabayashi, FOCS'05]

Approximation algorithm for M-minor-free graphs

(1) Relying on the previous Theorem, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of the Lemma on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$.

(4) If no solution is found, output the whole graph G.

This algorithm provides an $O(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$.

The running time of the algorithm is polynomial in n, since in step (2), for each G_i, the dynamic programming algorithm runs in $O((d+1)^{t_i}(t_i+1)^{d^2} n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

Approximation algorithm for M-minor-free graphs

(1) Relying on the previous Theorem, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of the Lemma on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$.

(4) If no solution is found, output the whole graph G.

This algorithm provides an $\mathcal{O}(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$.

The running time of the algorithm is polynomial in n, since in step (2), for each G_i, the dynamic programming algorithm runs in $\mathcal{O}((d+1)^{t_i}(t_i+1)^{d^2} n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

Approximation algorithm for M-minor-free graphs

(1) Relying on the previous Theorem, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of the Lemma on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$.

(4) If no solution is found, output the whole graph G.

This algorithm provides an $\mathcal{O}(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$.

The running time of the algorithm is polynomial in n, since in step **(2)**, for each G_i, the dynamic programming algorithm runs in $\mathcal{O}((d+1)^{t_i}(t_i+1)^{d^2} n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

Approximation algorithm for M-minor-free graphs

(1) Relying on the previous Theorem, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of the Lemma on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$.

(4) If no solution is found, output the whole graph G.

This algorithm provides an $\mathcal{O}(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$.

The running time of the algorithm is polynomial in n, since in step **(2)**, for each G_i, the dynamic programming algorithm runs in $\mathcal{O}((d+1)^{t_i}(t_i+1)^{d^2} n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

Approximation algorithm for M-minor-free graphs

(1) Relying on the previous Theorem, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of the Lemma on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$.

(4) If no solution is found, output the whole graph G.

This algorithm provides an $\mathcal{O}(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$.

The running time of the algorithm is polynomial in n, since in step **(2)**, for each G_i, the dynamic programming algorithm runs in $\mathcal{O}((d+1)^{t_i}(t_i+1)^{d^2} n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

Approximation algorithm for M-minor-free graphs

(1) Relying on the previous Theorem, partition $V(G)$ in polynomial time into $\log n + 1$ sets $V_0, \ldots, V_{\log n}$ such that any $\log n$ of the sets induce a graph of treewidth at most $c_M \log n$, where c_M is a constant depending only on the excluded graph M.

(2) Run the dynamic programming algorithm of the Lemma on all the subgraphs $G_i = G[V \setminus V_i]$ of $\log n$ sets, $i = 0, \ldots, \log n$.

(3) This procedure finds all the solutions of size at most $\log n$.

(4) If no solution is found, output the whole graph G.

This algorithm provides an $\mathcal{O}(n/\log n)$-approximation for MSMD_d in minor-free graphs, for all $d \geq 3$.

The running time of the algorithm is polynomial in n, since in step **(2)**, for each G_i, the dynamic programming algorithm runs in $\mathcal{O}((d+1)^{t_i}(t_i+1)^{d^2} n)$ time, where t_i is the treewidth of G_i, which is at most $c_M \log n$.

3- Dual Degree-Dense *k*-Subgraph (DDD*k*S)

Definition of the problem + results

- **DUAL DEGREE-DENSE *k*-SUBGRAPH (DDD*k*S)**:

  Input: an undirected graph $G = (V, E)$ and a positive integer k.

  Output: a subset $S \subseteq V$ with $|S| \leq k$, s.t. $\delta(G[S])$ is maximum.

- It is the natural *dual* version of the preceding problem.

- Our results:
  - Randomized $\mathcal{O}(\sqrt{n} \log n)$-approximation algorithm in general graphs.
  - Deterministic $\mathcal{O}(n^{\delta})$-approximation algorithm in general graphs, for some universal constant $\delta < 1/3$.

Definition of the problem + results

- **DUAL DEGREE-DENSE k-SUBGRAPH (DDDkS)**:

  Input: an undirected graph $G = (V, E)$ and a positive integer k.

  Output: a subset $S \subseteq V$ with $|S| \leq k$, s.t. $\delta(G[S])$ is maximum.

- It is the natural *dual* version of the preceding problem.

- Our results:
  - Randomized $\mathcal{O}(\sqrt{n} \log n)$-approximation algorithm in general graphs.
  - Deterministic $\mathcal{O}(n^{\delta})$-approximation algorithm in general graphs, for some universal constant $\delta < 1/3$.

Definition of the problem + results

- **DUAL DEGREE-DENSE k-SUBGRAPH (DDDkS)**:

  Input: an undirected graph $G = (V, E)$ and a positive integer k.

  Output: a subset $S \subseteq V$ with $|S| \leq k$, s.t. $\delta(G[S])$ is maximum.

- It is the natural *dual* version of the preceding problem.

- Our results:
  - Randomized $\mathcal{O}(\sqrt{n} \log n)$-approximation algorithm in general graphs.
  - Deterministic $\mathcal{O}(n^{\delta})$-approximation algorithm in general graphs, for some universal constant $\delta < 1/3$.

Definition of the problem + results

- **DUAL DEGREE-DENSE k-SUBGRAPH (DDDkS)**:

  Input: an undirected graph $G = (V, E)$ and a positive integer k.

  Output: a subset $S \subseteq V$ with $|S| \leq k$, s.t. $\delta(G[S])$ is maximum.

- It is the natural *dual* version of the preceding problem.

- Our results:
  - Randomized $\mathcal{O}(\sqrt{n} \log n)$-approximation algorithm in general graphs.
  - Deterministic $\mathcal{O}(n^{\delta})$-approximation algorithm in general graphs, for some universal constant $\delta < 1/3$.

Further Research

- **Problem 1:**
  - Approximation algorithms and hardness results in general graphs.
  - **Open:** closing the *huge* complexity gap of $MDBCS_d$, $d \geq 2$.
- **Problem 2:**
  - Hardness results and an approximation algorithm in minor-free graphs.
  - **Open:** finding approximation algorithms in general graphs for $MSMD_d$, $d \geq 3$.
- **Problem 3:**
  - Approximation algorithms in general graphs.
  - **Open:** hardness results for DDDkS, $k \geq 3$.

Further Research

- **Problem 1:**
  - Approximation algorithms and hardness results in general graphs.
  - **Open:** closing the *huge* complexity gap of MDBCS_d, $d \geq 2$.
- **Problem 2:**
  - Hardness results and an approximation algorithm in minor-free graphs.
  - **Open:** finding approximation algorithms in general graphs for MSMD_d, $d \geq 3$.
- **Problem 3:**
  - Approximation algorithms in general graphs.
  - **Open:** hardness results for DDDkS, $k \geq 3$.

Further Research

- **Problem 1:**
  - Approximation algorithms and hardness results in general graphs.
  - **Open:** closing the *huge* complexity gap of MDBCS_d, $d \geq 2$.
- **Problem 2:**
  - Hardness results and an approximation algorithm in minor-free graphs.
  - **Open:** finding approximation algorithms in general graphs for MSMD_d, $d \geq 3$.
- **Problem 3:**
  - Approximation algorithms in general graphs.
  - **Open:** hardness results for DDDkS, $k \geq 3$.

Further Research

- **Problem 1:**
  - Approximation algorithms and hardness results in general graphs.
  - **Open:** closing the *huge* complexity gap of MDBCS_d, $d \geq 2$.
- **Problem 2:**
  - Hardness results and an approximation algorithm in minor-free graphs.
  - **Open:** finding approximation algorithms in general graphs for MSMD_d, $d \geq 3$.
- **Problem 3:**
  - Approximation algorithms in general graphs.
  - **Open:** hardness results for DDDkS, $k \geq 3$.

Further Research

- **Problem 1:**
  - Approximation algorithms and hardness results in general graphs.
  - **Open:** closing the *huge* complexity gap of $MDBCS_d$, $d \geq 2$.
- **Problem 2:**
  - Hardness results and an approximation algorithm in minor-free graphs.
  - **Open:** finding approximation algorithms in general graphs for $MSMD_d$, $d \geq 3$.
- **Problem 3:**
  - Approximation algorithms in general graphs.
  - **Open:** hardness results for DDDkS, $k \geq 3$.

Further Research

- **Problem 1:**
  - Approximation algorithms and hardness results in general graphs.
  - **Open:** closing the *huge* complexity gap of MDBCS_d, $d \geq 2$.
- **Problem 2:**
  - Hardness results and an approximation algorithm in minor-free graphs.
  - **Open:** finding approximation algorithms in general graphs for MSMD_d, $d \geq 3$.
- **Problem 3:**
  - Approximation algorithms in general graphs.
  - **Open:** hardness results for DDDkS, $k \geq 3$.

Thanks!