Sum-Max Graph Partitioning Problem

Rémi Watrigant, Marin Bougeret, Rodolphe Giroudeau and Jean-Claude König

LIRMM, Montpellier, France

Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier

LIRMM

International Symposium on Combinatorial Optimization
Athens
April 17-21 2012

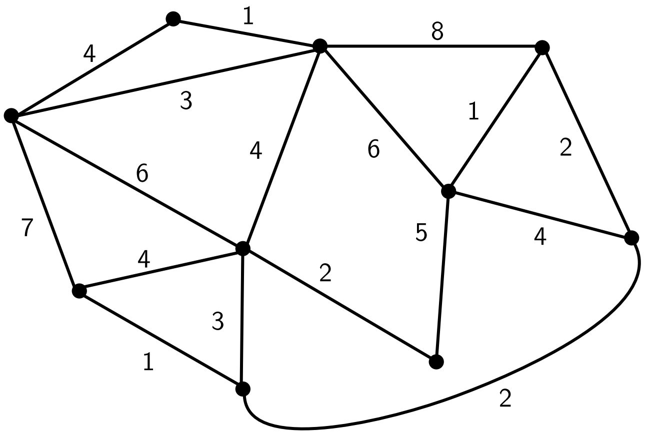
Contents

Description of the problem

Given a connected graph $G = (V, E)$ with weights on edges, and $k \leq |V|$

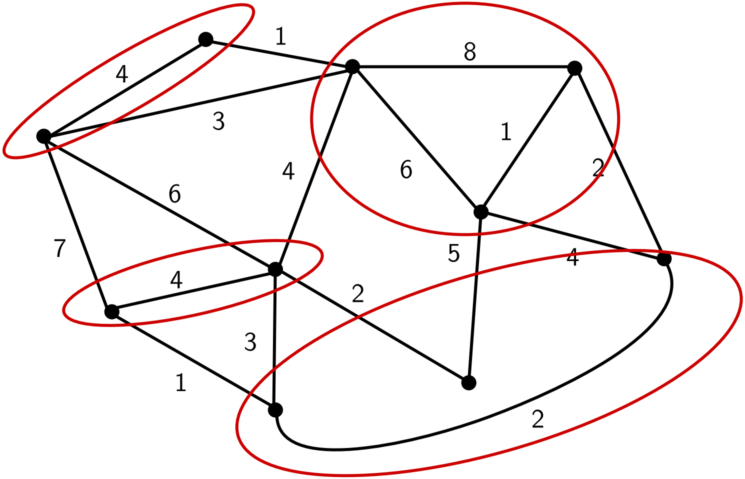
Description of the problem

Given a connected graph $G = (V, E)$ with weights on edges, and $k \leq |V|$
Example : k=4



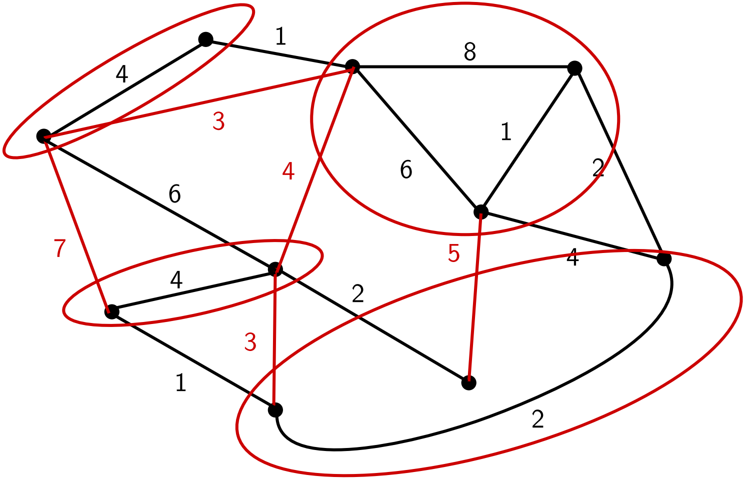
Description of the problem

Given a connected graph $G = (V, E)$ with weights on edges, and $k \leq |V|$
Example : k=4



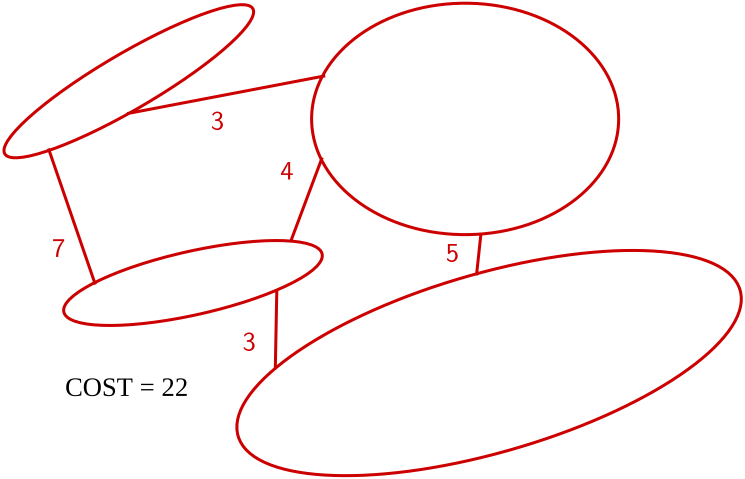
Description of the problem

Given a connected graph $G = (V, E)$ with weights on edges, and $k \leq |V|$
Example : k=4



Description of the problem

Given a connected graph $G = (V, E)$ with weights on edges, and $k \leq |V|$
Example : k=4



SUM-MAX GRAPH PARTITIONING

Input: a connected graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \in \mathbb{N}$
Output: a k-partition $(V_1, ..., V_k)$ of V

Goal: minimize $\displaystyle\sum_{\substack{i,j=1\\ i>j}}^{k} \max_{\substack{u \in V_i\\ v \in V_j}} w(u, v)$

SUM-MAX GRAPH PARTITIONING

Input: a connected graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \in \mathbb{N}$
Output: a k-partition $(V_1, ..., V_k)$ of V
Goal: minimize $\displaystyle\sum_{\substack{i,j=1 \\ i>j}}^{k} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v)$

In this talk:

Sum-Max Graph Partitioning

Input: a connected graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \in \mathbb{N}$
Output: a k-partition $(V_1, ..., V_k)$ of V
Goal: minimize $\sum_{\substack{i,j=1 \\ i>j}}^{k} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v)$

In this talk:

- hardness :$\mathcal{NP}$-hard, $W[1]$-hard w.r.t. k, not approximable within $O(n^{1-\epsilon})$ unless $\mathcal{P} = \mathcal{NP}$
  (even in the unweighted case)

SUM-MAX GRAPH PARTITIONING

Input: a connected graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \in \mathbb{N}$
Output: a k-partition $(V_1, ..., V_k)$ of V
Goal: minimize $\sum_{\substack{i,j=1 \\ i>j}}^{k} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v)$

In this talk:

- hardness :$\mathcal{NP}$-hard, $W[1]$-hard w.r.t. k, not approximable within $O(n^{1-\epsilon})$ unless $\mathcal{P} = \mathcal{NP}$
(even in the unweighted case)
- greedy $\frac{k}{2}$-approximation algorithm

SUM-MAX GRAPH PARTITIONING

Input: a connected graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \in \mathbb{N}$
Output: a k-partition $(V_1, ..., V_k)$ of V
Goal: minimize $\displaystyle\sum_{\substack{i,j=1\\ i>j}}^{k} \max_{\substack{u\in V_i\\ v\in V_j}} w(u, v)$

In this talk:

- hardness :$\mathcal{NP}$-hard, $W[1]$-hard w.r.t. k, not approximable within $O(n^{1-\epsilon})$ unless $\mathcal{P} = \mathcal{NP}$
  (even in the unweighted case)
- greedy $\frac{k}{2}$-approximation algorithm
- exact solutions ($k = 3$ and extensions)

Sum-Max Graph Partitioning

Input: a connected graph $G = (V, E)$, $w : E \to \mathbb{N}$, $k \in \mathbb{N}$
Output: a k-partition $(V_1, ..., V_k)$ of V
Goal: minimize $\sum_{\substack{i,j=1 \\ i>j}}^{k} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v)$

In this talk:

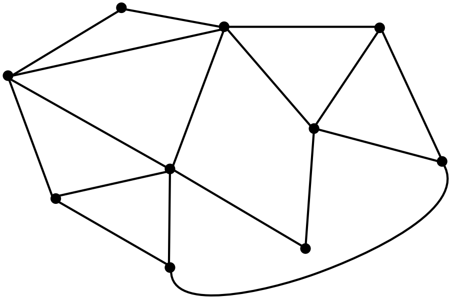
- hardness :$\mathcal{NP}$-hard, $W[1]$-hard w.r.t. k, not approximable within $O(n^{1-\epsilon})$ unless $\mathcal{P} = \mathcal{NP}$
  (even in the unweighted case)
- greedy $\frac{k}{2}$-approximation algorithm
- exact solutions ($k = 3$ and extensions)
- partial results and future work

Contents

Unweighted version of the problem

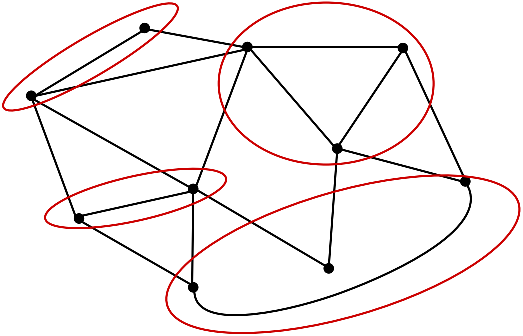
$w(e) = 1 \ \forall e \in E$

Example, $k = 4$



Unweighted version of the problem

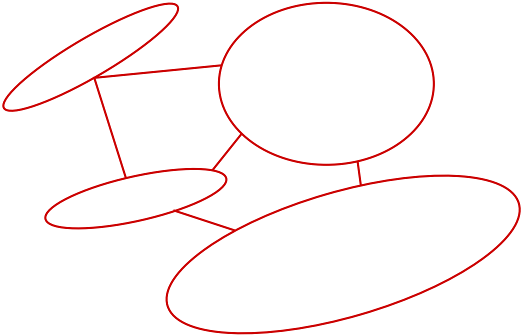
$w(e) = 1 \ \forall e \in E$ Example, $k = 4$



Unweighted version of the problem

$w(e) = 1 \ \forall e \in E$

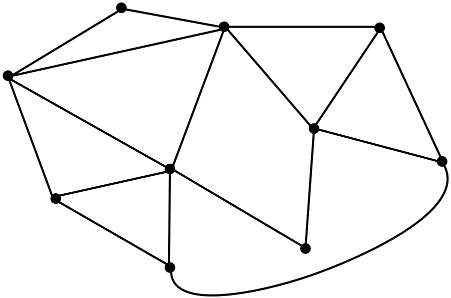
Example, $k = 4$



Unweighted version of the problem

$w(e) = 1 \ \forall e \in E$

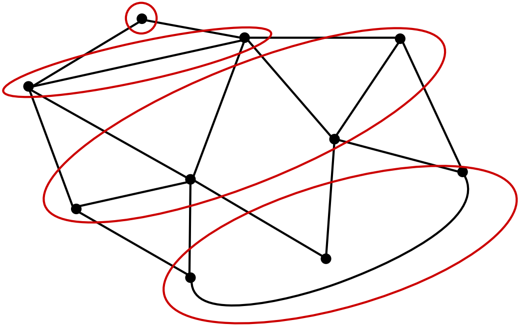
Example, $k = 4$



Unweighted version of the problem

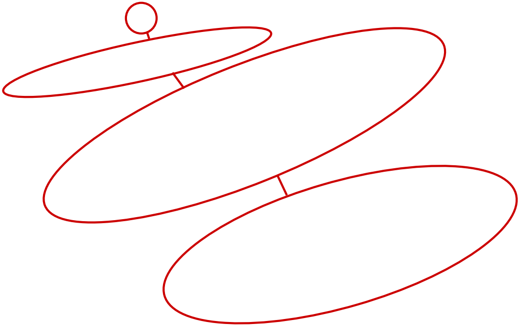
$w(e) = 1 \ \forall e \in E$

Example, $k = 4$



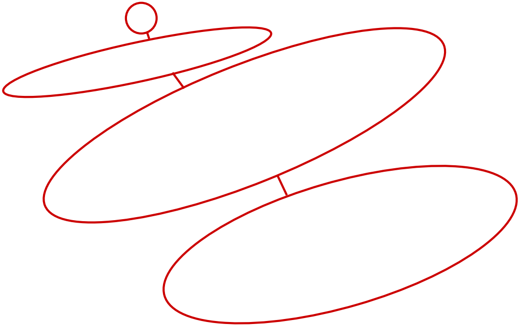
Unweighted version of the problem

$w(e) = 1 \; \forall e \in E$ Example, $k = 4$



Unweighted version of the problem

$w(e) = 1 \ \forall e \in E$ Example, $k = 4$



Property

For a connected unweighted graph, any optimal solution has a cost of at least $k-1$ (in this case the quotient graph is a tree)

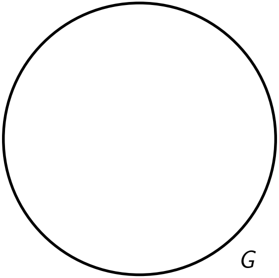
NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

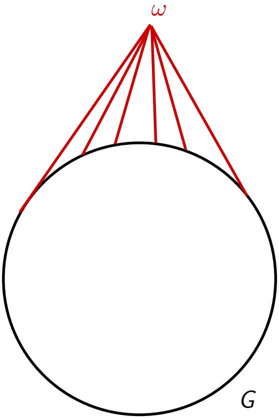
graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?



NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

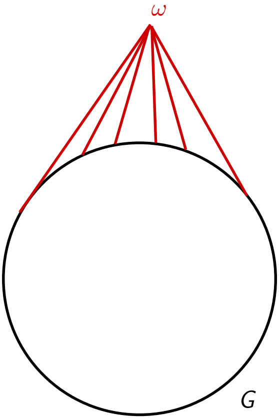
graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?



NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?

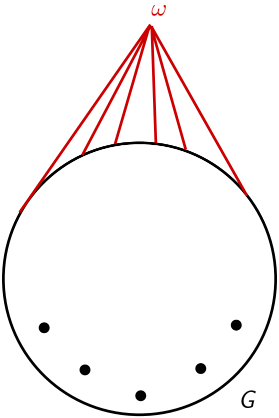


$\alpha(G) \geq k \Rightarrow G'$ has a $(k+1)$-partition of cost k

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?

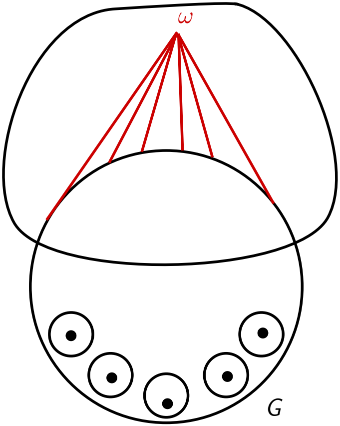


$\alpha(G) \geq k \Rightarrow G'$ has a $(k+1)$-partition of cost k

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?

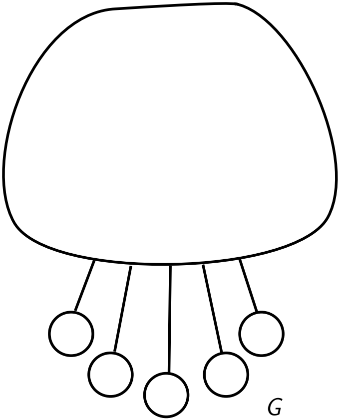


$\alpha(G) \geq k \Rightarrow G'$ has a $(k+1)$-partition of cost k

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?



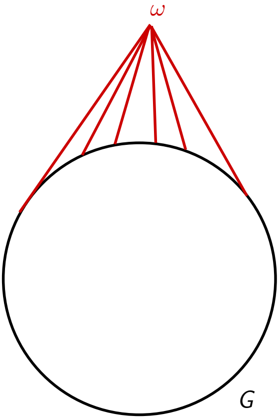
$\alpha(G) \geq k \Rightarrow G'$ has a $(k+1)$-partition of cost k

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$ $\rightarrow$ $G \cup \{\omega\}$, $(k+1)$-partition

$\alpha(G) \geq k$? $(k+1)$-partition of cost k ?



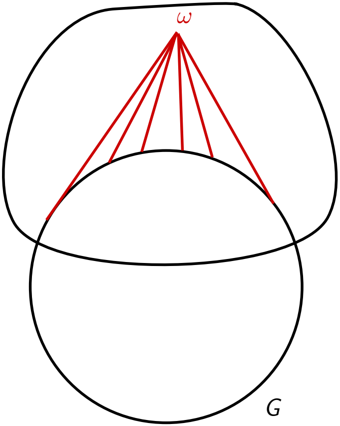
$\alpha(G) < k \Rightarrow$ any $(k+1)$-partition of G' has a cost $\geq k+1$

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$ $\rightarrow$ $G \cup \{\omega\}$, $(k+1)$-partition

$\alpha(G) \geq k$? $(k+1)$-partition of cost k ?



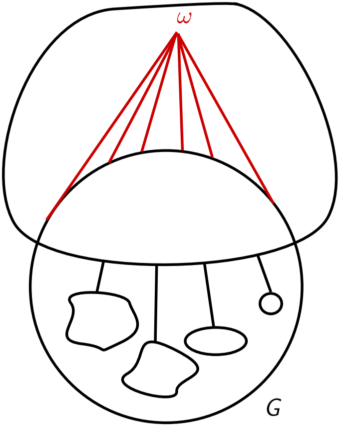
$\alpha(G) < k \Rightarrow$ any $(k+1)$-partition of G' has a cost $\geq k+1$

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$ $\rightarrow$ $G \cup \{\omega\}$, $(k+1)$-partition

$\alpha(G) \geq k$? $(k+1)$-partition of cost k ?



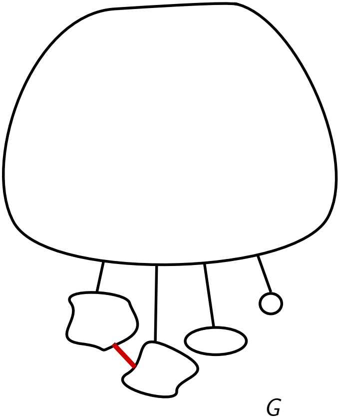
$\alpha(G) < k \Rightarrow$ any $(k+1)$-partition of G' has a cost $\geq k+1$

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$ $\rightarrow$ $G \cup \{\omega\}$, $(k+1)$-partition

$\alpha(G) \geq k$? $(k+1)$-partition of cost k ?

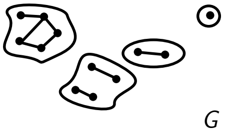


$\alpha(G) < k \Rightarrow$ any $(k+1)$-partition of G' has a cost $\geq k + 1$

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$	$\rightarrow$	$G \cup \{\omega\}$, $(k+1)$-partition
$\alpha(G) \geq k$?		$(k+1)$-partition of cost k ?



$\alpha(G) < k \Rightarrow$ any $(k+1)$-partition of G' has a cost $\geq k+1$

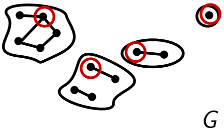
NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

graph G, $k \in \mathbb{N}$ $\rightarrow$ $G \cup \{\omega\}$, $(k+1)$-partition

$\alpha(G) \geq k$? $(k+1)$-partition of cost k ?

independent set of size k



$\alpha(G) < k \Rightarrow$ any $(k+1)$-partition of G' has a cost $\geq k+1$

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:
Conclusion:

- $\alpha(G) \geq k \Rightarrow OPT(G') \leq k$
- $\alpha(G) < k \Rightarrow OPT(G') > k$

Theorem

SUM-MAX GRAPH PARTITIONING is $\mathcal{NP}$-hard, and even $W[1]$-hard for the parameter k

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:
Conclusion:

- $\alpha(G) \geq k \Rightarrow OPT(G') \leq k$
- $\alpha(G) < k \Rightarrow OPT(G') > k$

Theorem

SUM-MAX GRAPH PARTITIONING is $\mathcal{NP}$-hard, and even $W[1]$-hard for the parameter k

We can also prove a gap preserving reduction :

- $\alpha(G) \geq k \Rightarrow OPT(G') \leq k$
- $\alpha(G) < r.k \Rightarrow OPT(G') > \frac{1}{2r}k$ for all $r << 1$

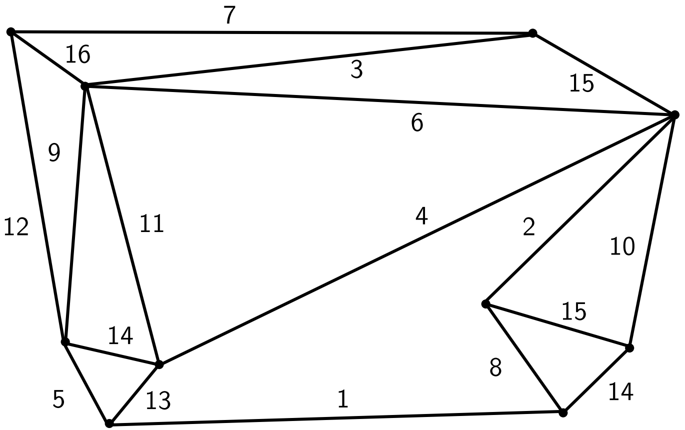
Theorem

SUM-MAX GRAPH PARTITIONING is $O(n^{1-\epsilon})$ non-approximable unless $\mathcal{P} = \mathcal{NP}$

Contents

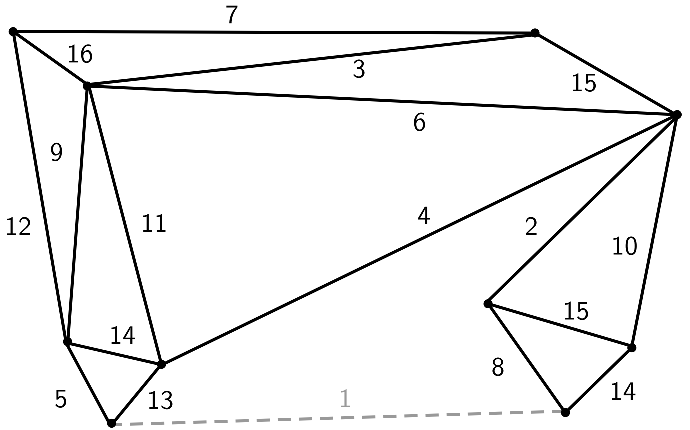
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



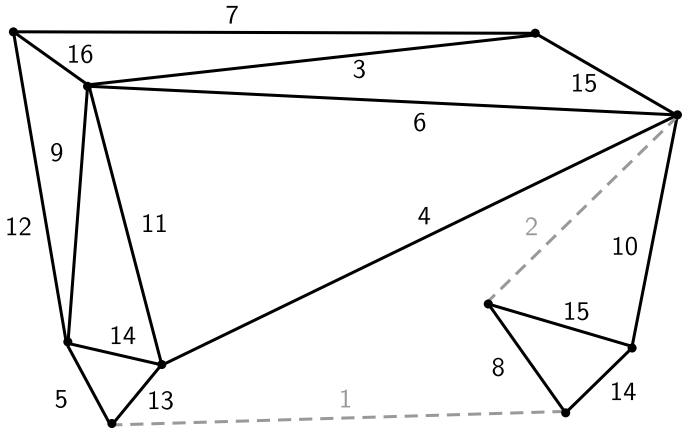
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



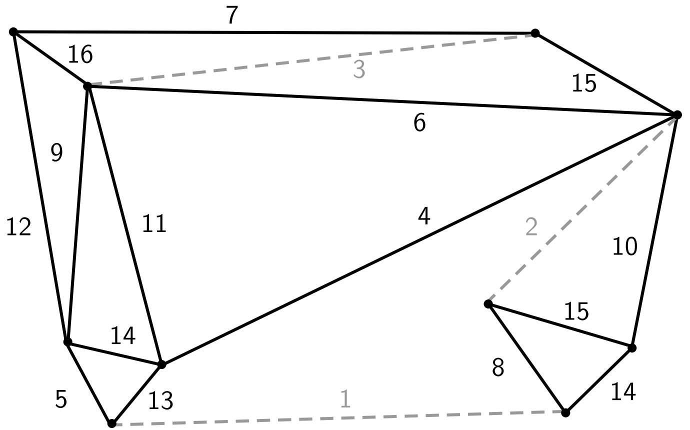
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



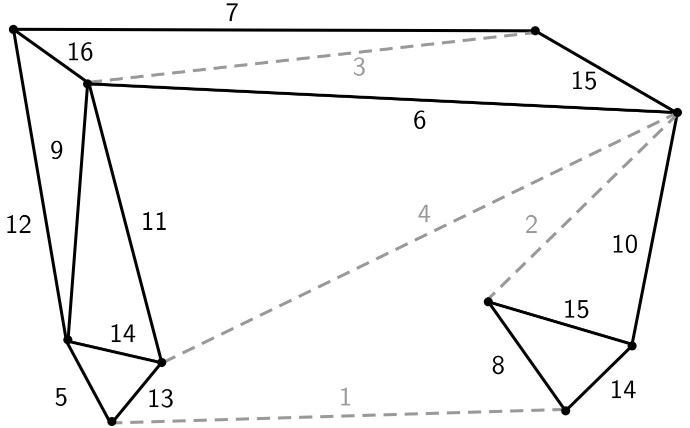
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



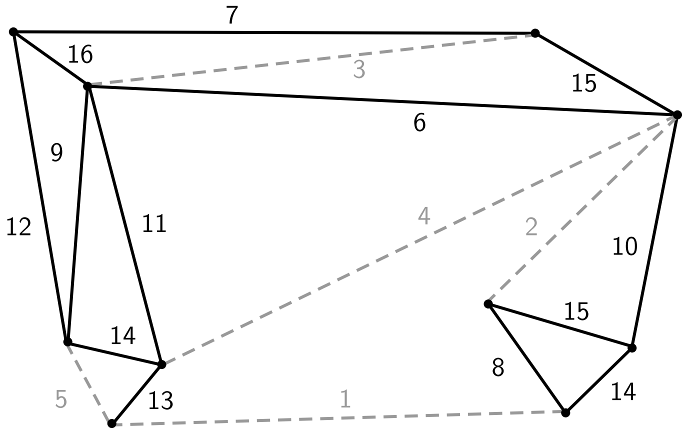
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



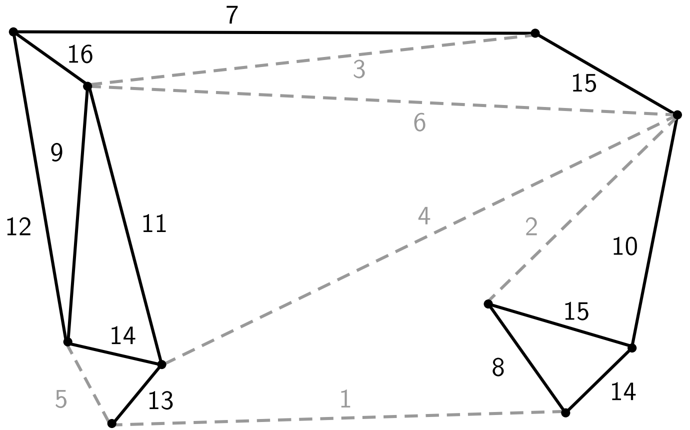
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



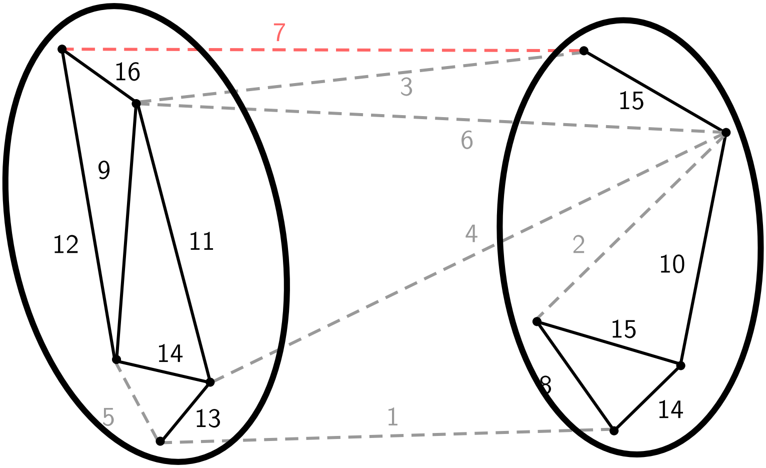
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



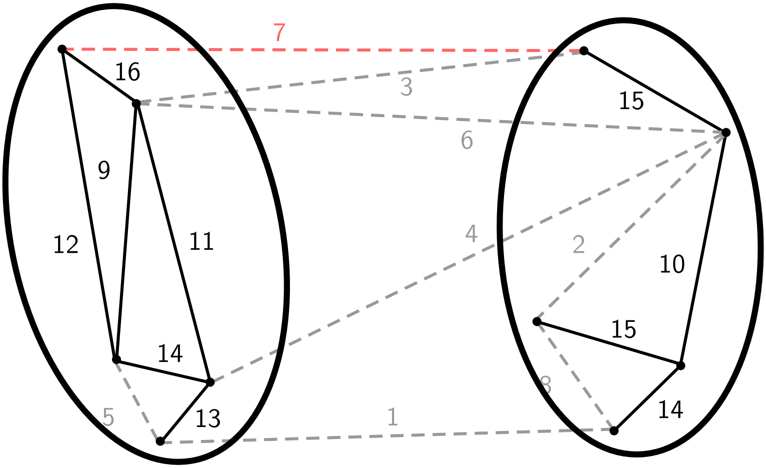
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



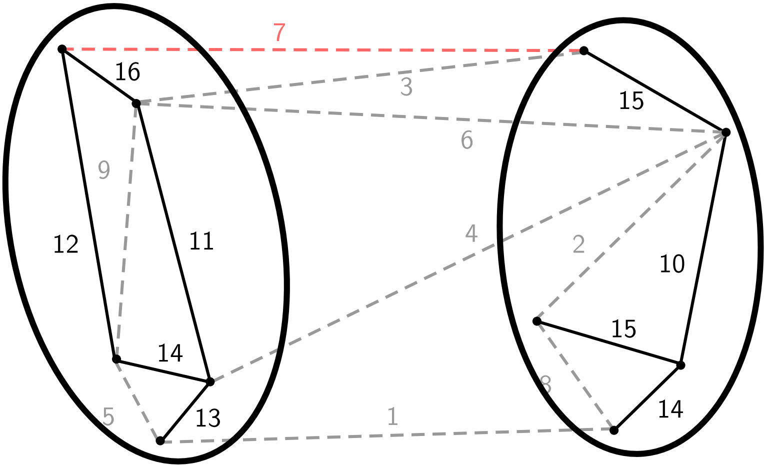
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



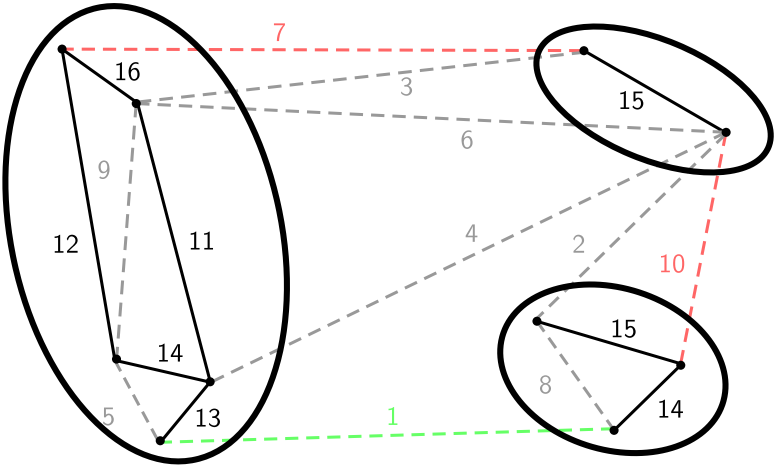
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



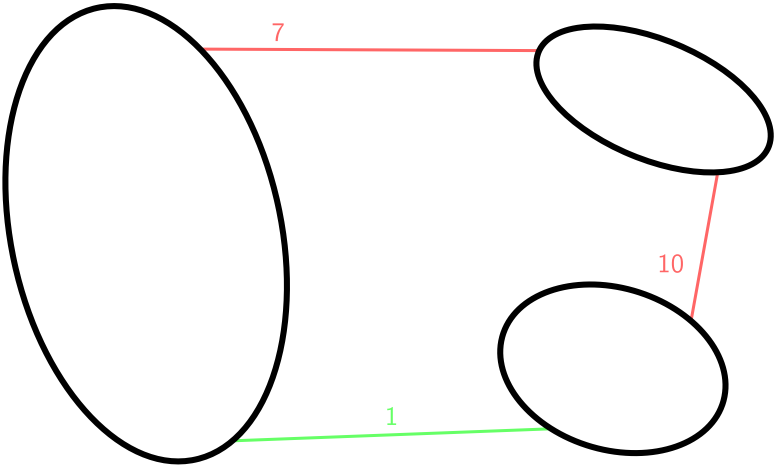
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



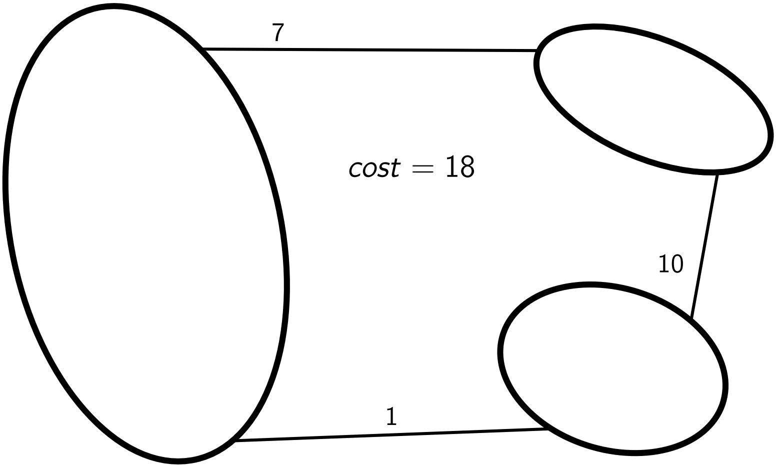
Example, $k = 3$

Algorithm = remove the lightest edge of G until G has k connected components



Example, $k = 3$

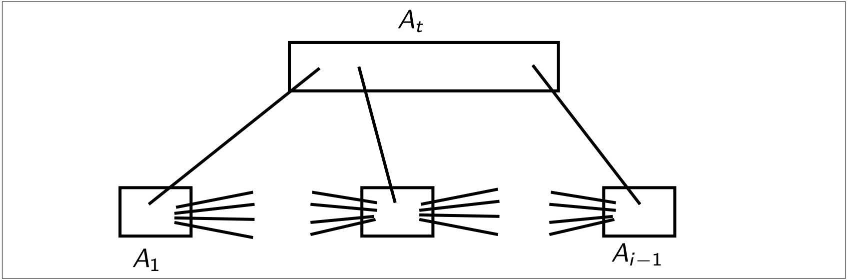
Algorithm = remove the lightest edge of G until G has k connected components



Algorithm:

Algorithm:

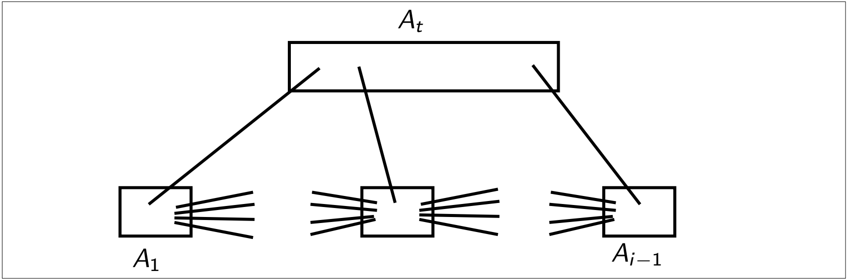
For i from 1 to $k-1$ do:



Algorithm:

For i from 1 to $k-1$ do:

while G has i connected components do:

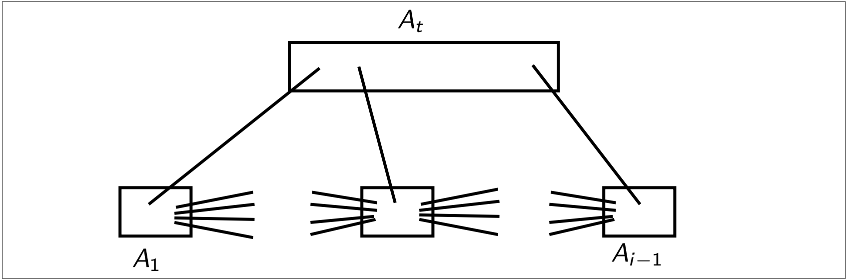


Algorithm:

For i from 1 to $k-1$ do:

    while G has i connected components do:

        remove the lightest edge in G



Algorithm:

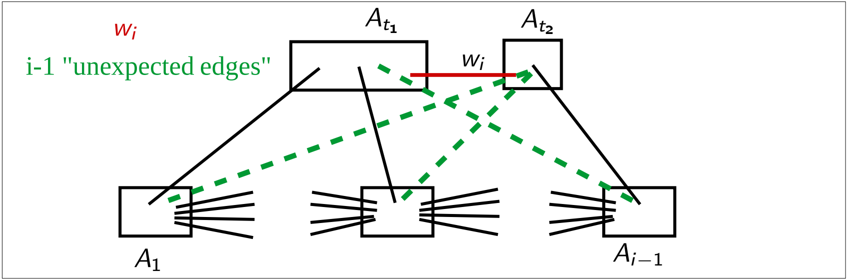
For i from 1 to $k-1$ do:

    while G has i connected components do:

        remove the lightest edge in G

    end while. *// let w_i be the weight of the last removed edge*

end for



Algorithm:

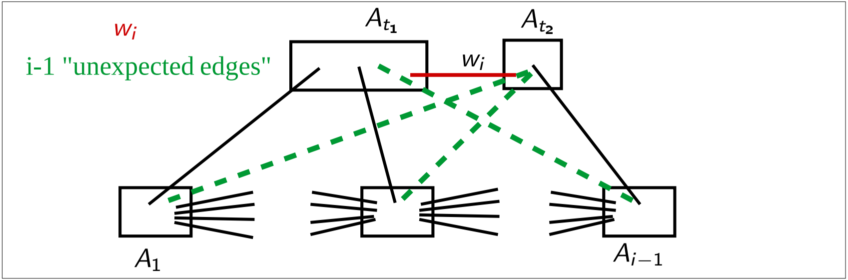
For i from 1 to $k-1$ do:

 while G has i connected components do:

  remove the lightest edge in G

 end while. *// let w_i be the weight of the last removed edge*

end for



At the end:

$$\text{Solution value} = \sum_{i=1}^{k-1} w_i + \sum \text{unexpected edges}$$

Algorithm:

For i from 1 to $k-1$ do:
  while G has i connected components do:
    remove the lightest edge in G
  end while. *// let w_i be the weight of the last removed edge*
end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Proof by induction over i:

Algorithm:

For i from 1 to $k-1$ do:

    while G has i connected components do:

        remove the lightest edge in G

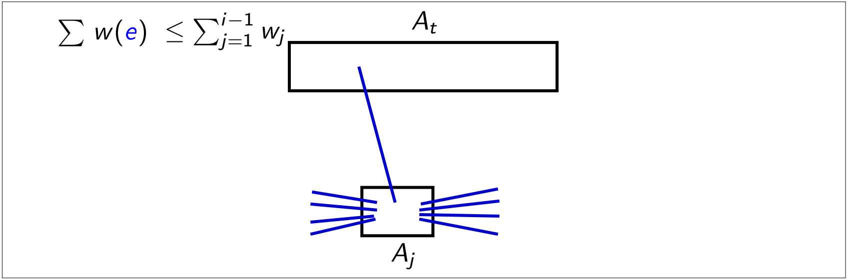
    end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Proof by induction over i:



Algorithm:

For i from 1 to $k-1$ do:

while G has i connected components do:

remove the lightest edge in G

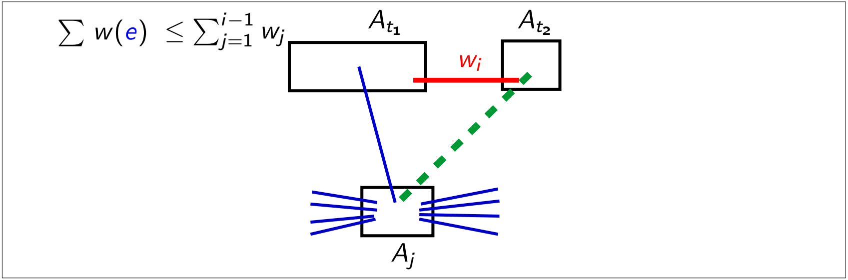
end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Proof by induction over i:



Algorithm:

For i from 1 to $k-1$ do:

    while G has i connected components do:

        remove the lightest edge in G

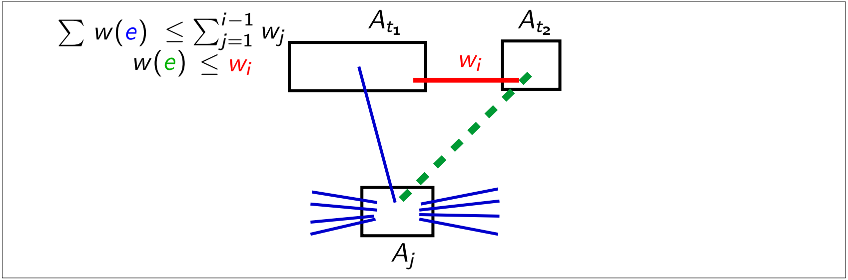
    end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Proof by induction over i:



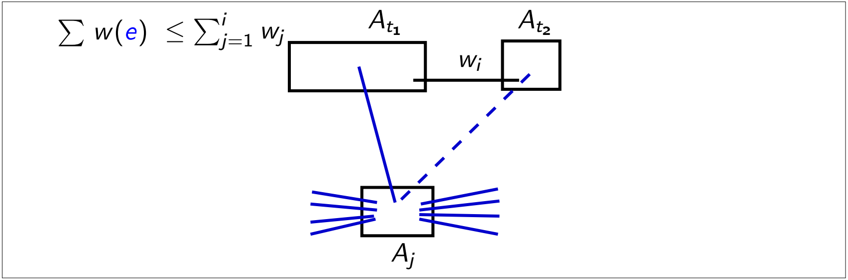
Algorithm:

For i from 1 to $k-1$ do:
    while G has i connected components do:
        remove the lightest edge in G
    end while. *// let w_i be the weight of the last removed edge*
end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Proof by induction over i:



Algorithm:

For i from 1 to $k-1$ do:

  while G has i connected components do:

    remove the lightest edge in G

  end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Thus: $$\mathcal{A} \leq \frac{k}{2} \sum_{j=1}^{k-1} w_j$$

Algorithm:

For i from 1 to $k-1$ do:

    while G has i connected components do:

        remove the lightest edge in G

    end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Thus: $\mathcal{A} \leq \frac{k}{2} \sum_{j=1}^{k-1} w_j$

Lemma 2

$$\sum_{j=1}^{k-1} w_j \leq OPT$$

Algorithm:

For i from 1 to $k-1$ do:

while G has i connected components do:

remove the lightest edge in G

end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

At each step i: sum of edges of maximum weight outgoing from each cluster is smaller than $\sum_{j=1}^{i-1} w_j$

Thus: $\mathcal{A} \leq \frac{k}{2} \sum_{j=1}^{k-1} w_j$

Lemma 2

$$\sum_{j=1}^{k-1} w_j \leq OPT$$

$$\Rightarrow \mathcal{A} \leq \frac{k}{2} OPT$$

(can be improved using the gap between edge weights)

Contents

Polynomial algorithm for $k = 3$

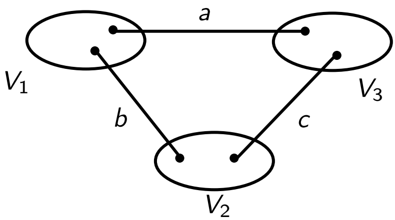
Idea of the algorithm:

Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)

suppose that $a \leq b \leq c$

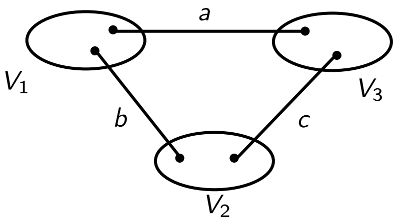


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$

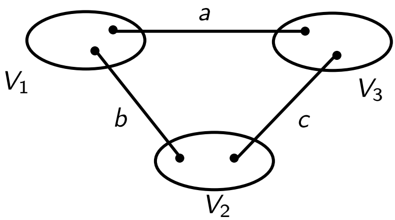


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$

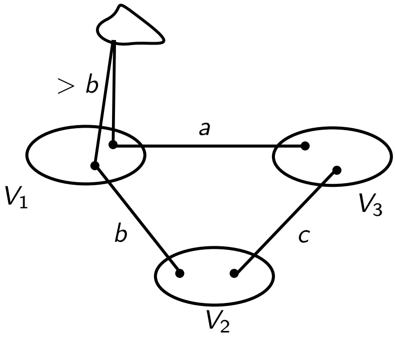


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$

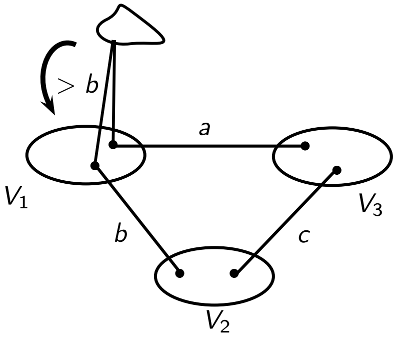


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$

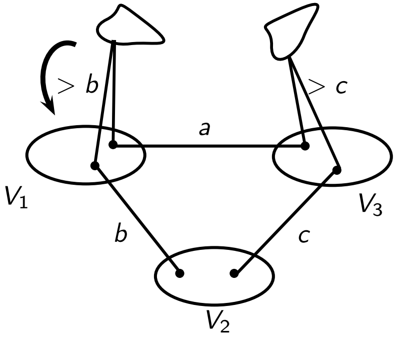


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$

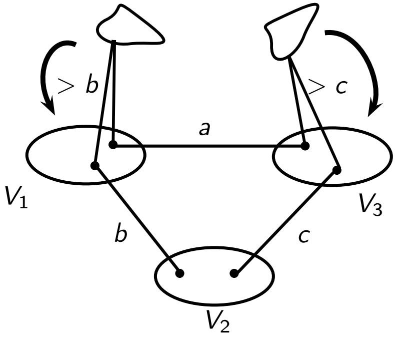


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$

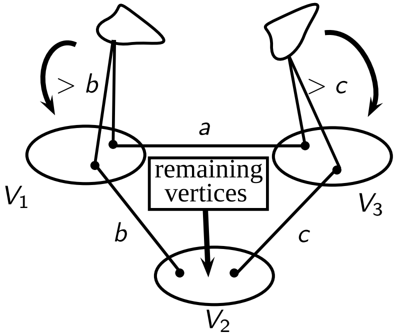


Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

suppose that $a \leq b \leq c$



Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

Extension to all fixed k (**unweighted version**):

Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

Extension to all fixed k (**unweighted version**):

- enumerate all "pattern graphs" to match to ($\equiv$ quotient graph of an optimal solution) in $O(m^{k^2})$ time

Polynomial algorithm for $k = 3$

Idea of the algorithm:

- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

Extension to all fixed k (**unweighted version**):

- enumerate all "pattern graphs" to match to ($\equiv$ quotient graph of an optimal solution) in $O(m^{k^2})$ time
- arrange all remaining vertices <u>without increasing the solution value</u>

Polynomial algorithm for $k = 3$

Idea of the algorithm:

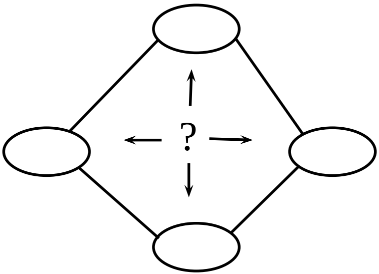
- guess the edges of maximum weight between the 3 clusters (in $O(m^3)$ time)
- arrange all remaining vertices without increasing the solution value

Extension to all fixed k (**unweighted version**):

- enumerate all "pattern graphs" to match to ($\equiv$ quotient graph of an optimal solution) in $O(m^{k^2})$ time
- arrange all remaining vertices <u>without increasing the solution value</u>

arranging vertices is $\mathcal{NP}$-complete if the quotient graph is C_4

(reduction from 3-COLORING)



Contents

Conclusion, future work

About the unweighted version of the problem:

- no $O(poly(n))$ algorithm, no $O(f(k)poly(n))$ algorithm...

Conclusion, future work

About the unweighted version of the problem:

- no $O(poly(n))$ algorithm, no $O(f(k)poly(n))$ algorithm...
  what about a $O(n^{f(k)})$ algorithm ? (is the problem in XP ?)

Conclusion, future work

About the unweighted version of the problem:

- no $O(poly(n))$ algorithm, no $O(f(k)poly(n))$ algorithm...
  what about a $O(n^{f(k)})$ algorithm ? (is the problem in XP ?)
- restricted graph classes :

Conclusion, future work

About the unweighted version of the problem:

- no $O(poly(n))$ algorithm, no $O(f(k)poly(n))$ algorithm... what about a $O(n^{f(k)})$ algorithm ? (is the problem in XP ?)
- restricted graph classes :

graph class	**negative result**	**positive result**
general graphs	$O(n^{1-\epsilon})$ inapprox.	$\frac{k}{2}$-approx
split graphs	$\mathcal{NP}$-complete	FPT (k), 2-approx
interval graphs	?	$O(n^{2k})$ exact algorithm
planar graphs	?	FPT (linear kernel)
chordal graphs	$\mathcal{NP}$-complete	?

Conclusion, future work

About the unweighted version of the problem:

- no $O(poly(n))$ algorithm, no $O(f(k)poly(n))$ algorithm...
  what about a $O(n^{f(k)})$ algorithm ? (is the problem in XP ?)
- restricted graph classes :

graph class	**negative result**	**positive result**
general graphs	$O(n^{1-\epsilon})$ inapprox.	$\frac{k}{2}$-approx
split graphs	$\mathcal{NP}$-complete	FPT (k), 2-approx
interval graphs	?	$O(n^{2k})$ exact algorithm
planar graphs	?	FPT (linear kernel)
chordal graphs	$\mathcal{NP}$-complete	?

- links with graph homomorphisms, notion of "**compaction**" [N. Vikas, P. Hell]
  our unweighted problem $\equiv$ compaction to a graph with few edges

Conclusion, future work

About the unweighted version of the problem:

- no $O(poly(n))$ algorithm, no $O(f(k)poly(n))$ algorithm...
  what about a $O(n^{f(k)})$ algorithm ? (is the problem in XP ?)
- restricted graph classes :

graph class	**negative result**	**positive result**
general graphs	$O(n^{1-\epsilon})$ inapprox.	$\frac{k}{2}$-approx
split graphs	$\mathcal{NP}$-complete	FPT (k), 2-approx
interval graphs	?	$O(n^{2k})$ exact algorithm
planar graphs	?	FPT (linear kernel)
chordal graphs	$\mathcal{NP}$-complete	?

- links with graph homomorphisms, notion of "**compaction**" [N. Vikas, P. Hell]
  our unweighted problem $\equiv$ compaction to a graph with few edges
- applications in software engineering : adding/relaxing constraints
  - constraints on cluster sizes
  - number of clusters not fixed

Thank you for your attention!