On the approximability of the 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 Workshop on Approximation, Parameterized and EXact algorithms
February 28-29, 2012, Paris, France

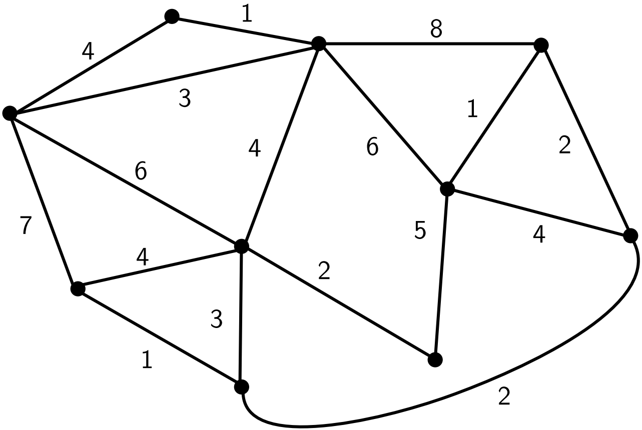
Contents

Description of the problem

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

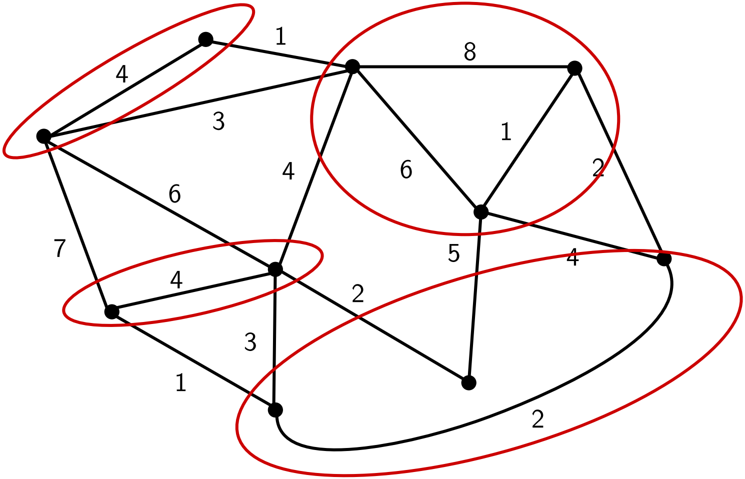
Description of the problem

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



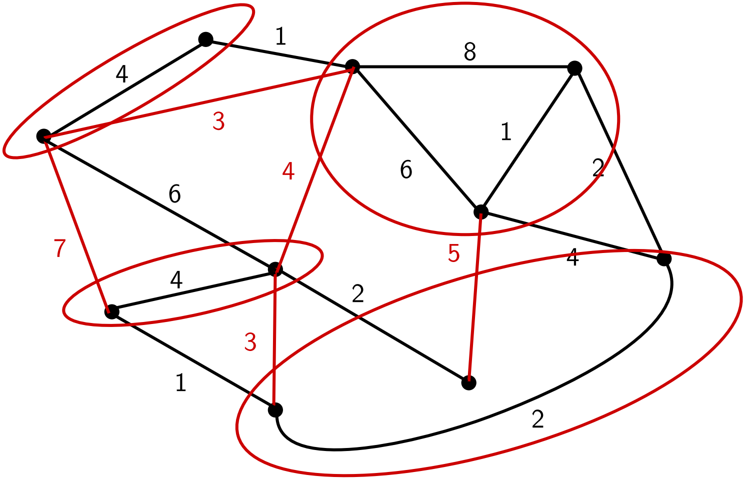
Description of the problem

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



Description of the problem

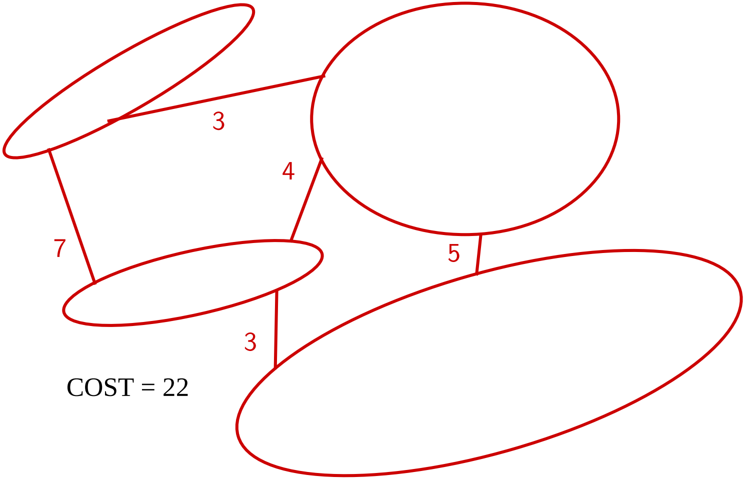
Given a connected graph $G = (V, E)$ with weights on edges, and $k \leq |V|$
Example : k=4



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

SUM-MAX GRAPH PARTITIONING

Input: a connected graph $G = (V, E)$, $w : E \rightarrow \mathbb{N}$, $k \in \mathbb{N}$
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:

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:

- simple $\frac{k}{2}$-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 $\sum_{\substack{i,j=1 \\ i>j}}^{k} \max_{\substack{u \in V_i \\ v \in V_j}} w(u, v)$

In this talk:

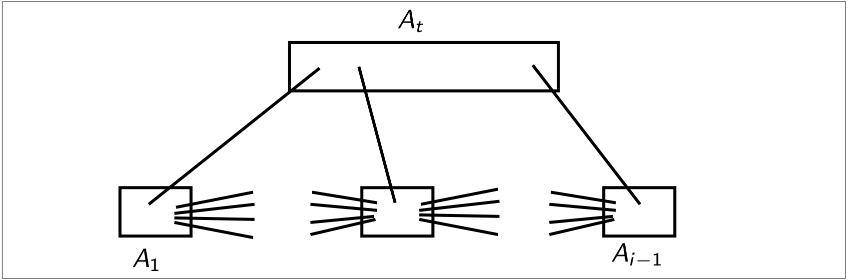
- simple $\frac{k}{2}$-approximation algorithm
- cannot be approximated with a factor in $O(n^{1-\epsilon})$ (unless $\mathcal{P} = \mathcal{NP}$) (and $\mathcal{NP}$-hardness, $W[1]$-hardness with parameter k)

Contents

Algorithm:

Algorithm:

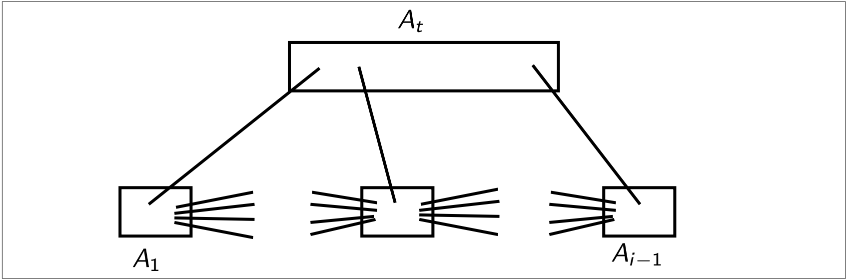
For i from 1 to $k-1$ do:



Algorithm:

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

while G has i connected components do:

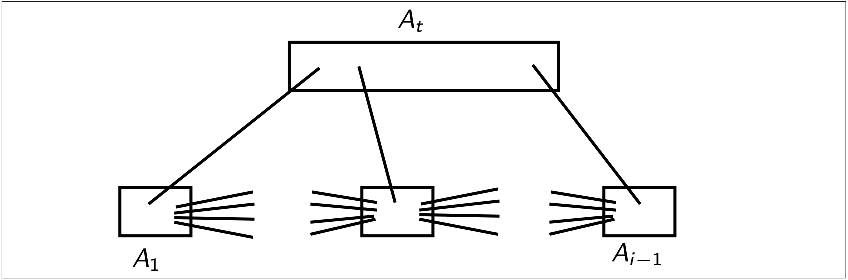


Algorithm:

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

while G has i connected components do:

remove the lightest edge in G



Algorithm:

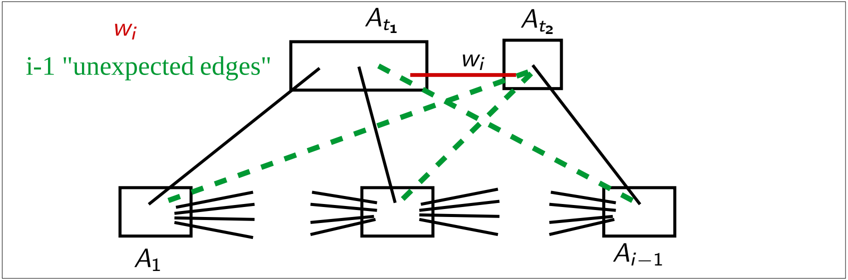
For i from 1 to $k-1$ do:

    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:

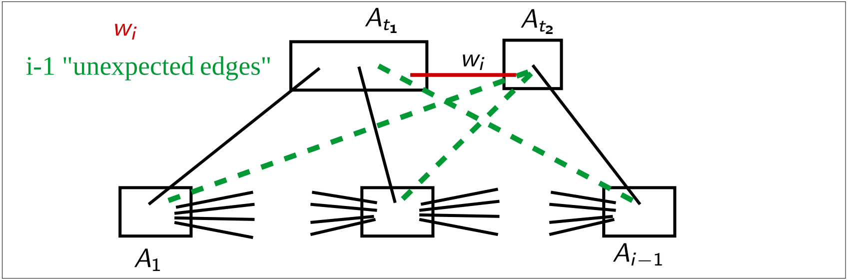
For i from 1 to $k-1$ do:

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 bounded by above by $\sum_{j=1}^{i-1} w_j$

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

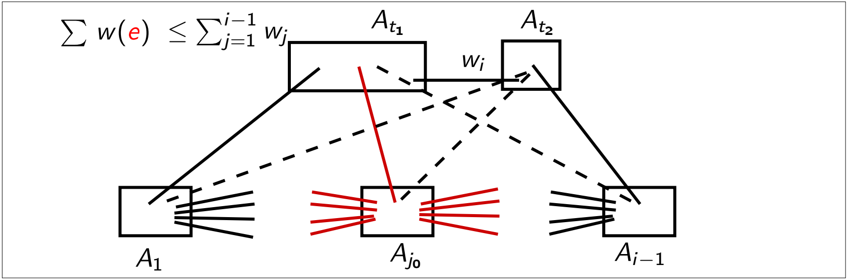
  end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

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

Proof by induction over i:

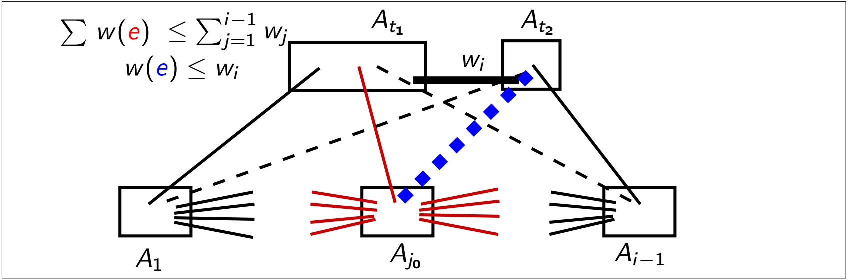


Algorithm:
For i from 1 to $k-1$ do:
    while G has i connected components do:
        remove the lightest edge in G
    end while. *// let w_i be the weight of the last removed edge*
end for

Lemma 1

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

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

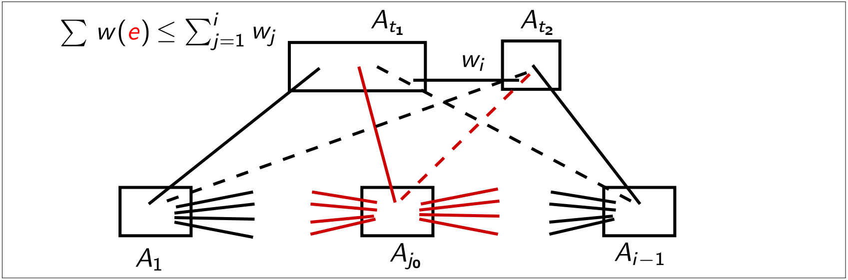
  end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

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

Proof by induction over i:

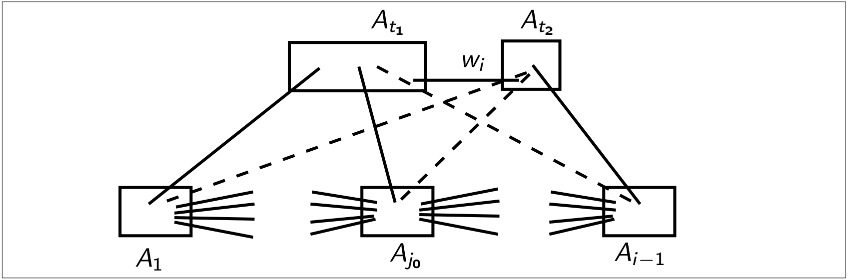


Algorithm:
For i from 1 to $k-1$ do:
    while G has i connected components do:
        remove the lightest edge in G
    end while. *// let w_i be the weight of the last removed edge*
end for

Lemma 1

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

Proof by induction over i:

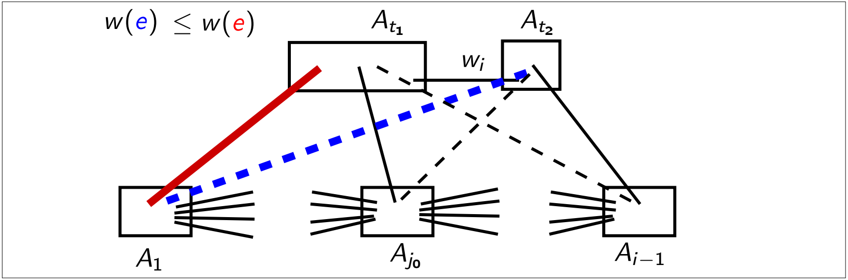


Algorithm:
For i from 1 to $k-1$ do:
  while G has i connected components do:
    remove the lightest edge in G
  end while. *// let w_i be the weight of the last removed edge*
end for

Lemma 1

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

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

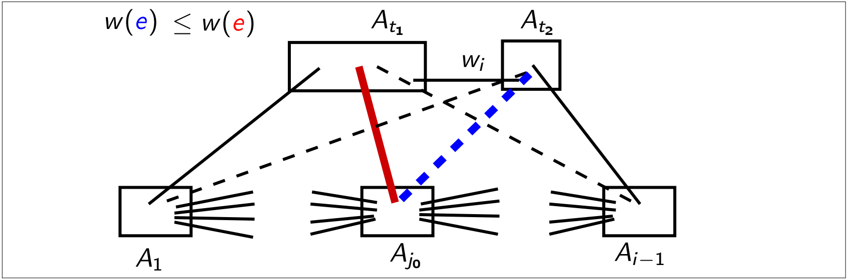
    end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

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

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

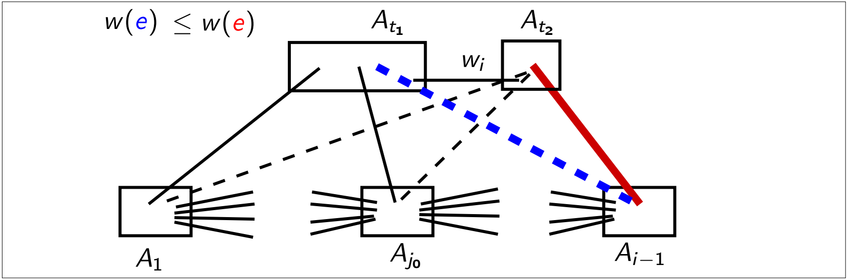
  end while. *// let w_i be the weight of the last removed edge*

end for

Lemma 1

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

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 bounded by above by $\sum_{j=1}^{i-1} w_j$

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 bounded by above by $\sum_{j=1}^{i-1} w_j$

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 bounded by above by $\sum_{j=1}^{i-1} w_j$

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

Lemma 2

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

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

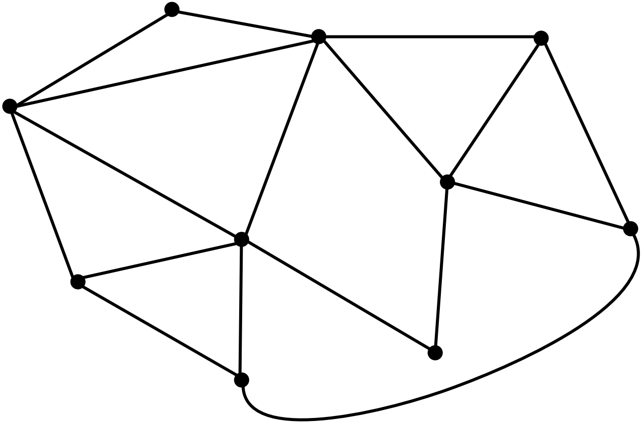
(can be improved using the gap between edge weights)

Contents

Unweighted version of the problem

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

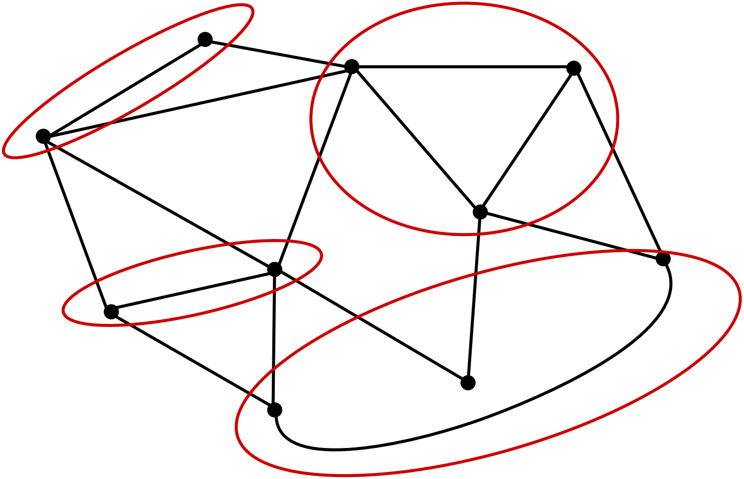
Example, $k = 4$



Unweighted version of the problem

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

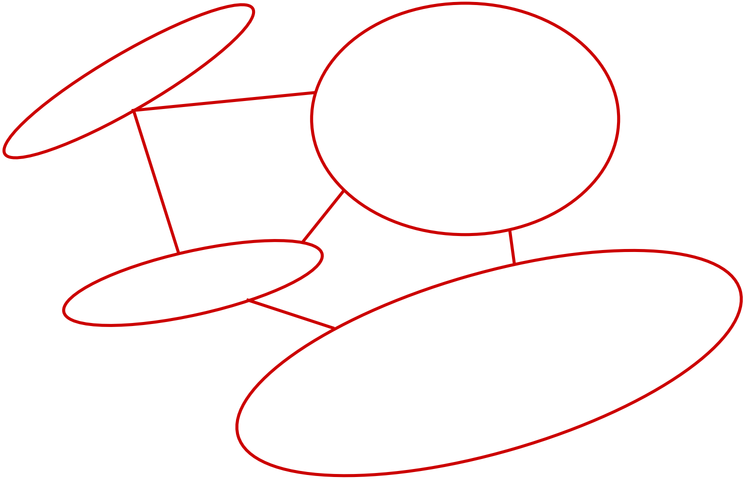
Example, $k = 4$



Unweighted version of the problem

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

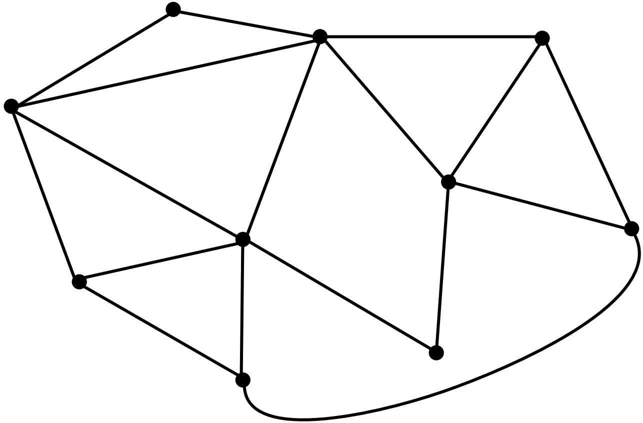
Example, $k = 4$



Unweighted version of the problem

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

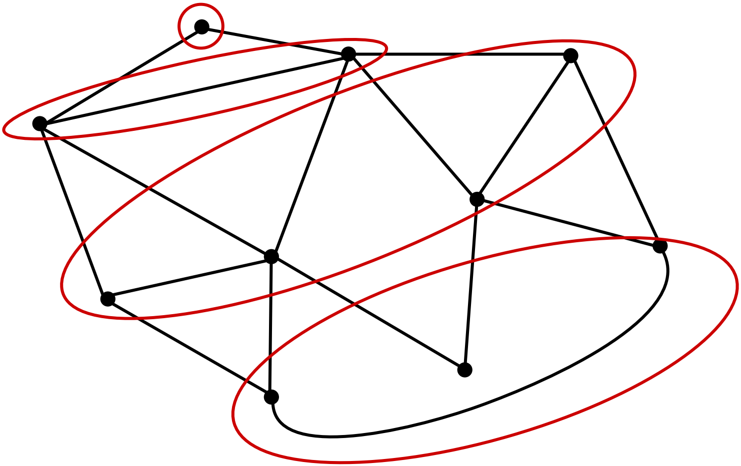
Example, $k = 4$



Unweighted version of the problem

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

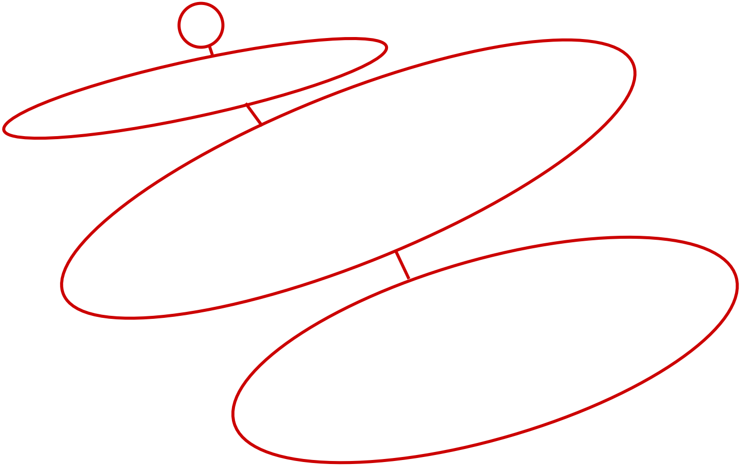
Example, $k = 4$



Unweighted version of the problem

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

Example, $k = 4$

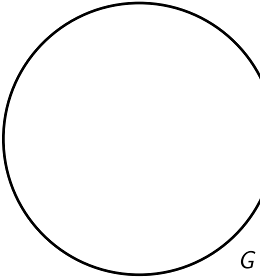


NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

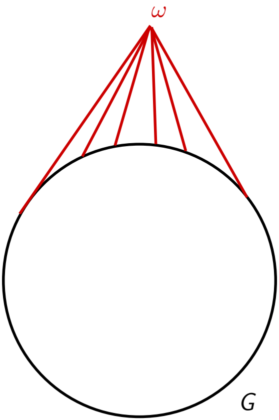
NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:



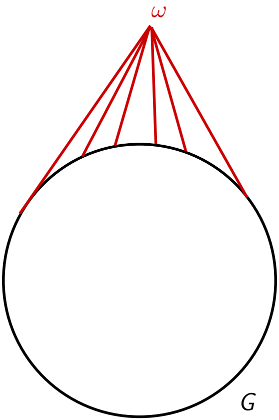
NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:



NP, $W[1]$ hardnesses, inapproximability

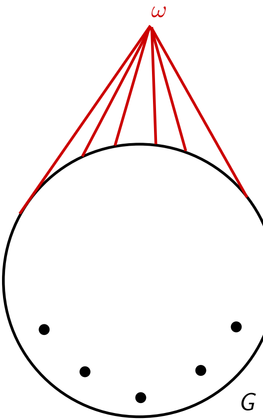
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

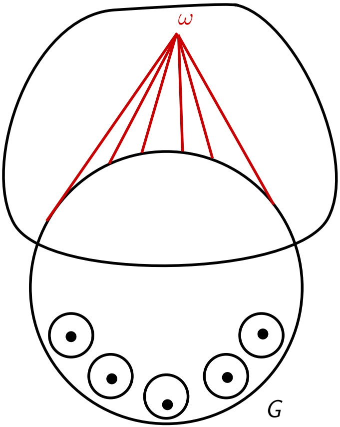
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

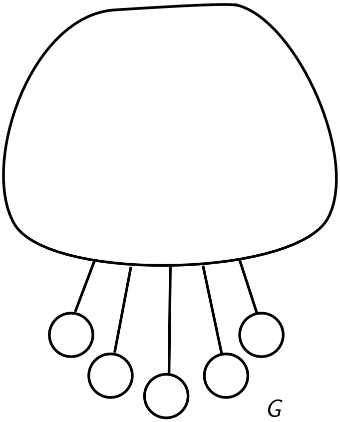
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

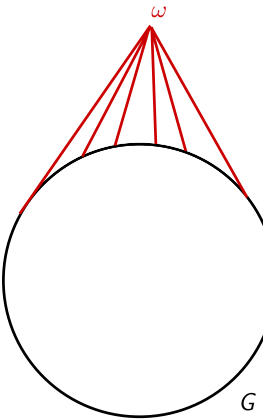
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

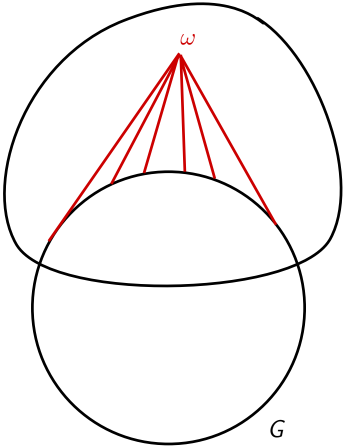
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

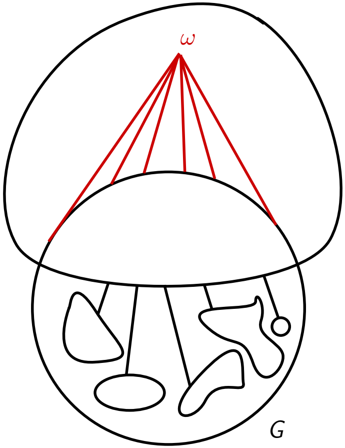
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

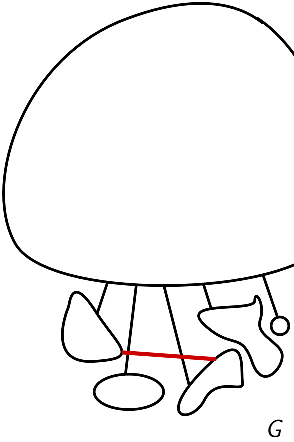
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

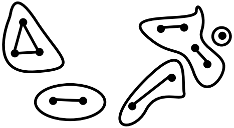
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

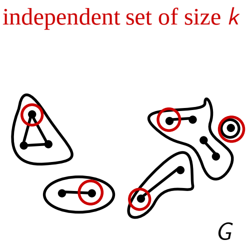
Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:



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

NP, $W[1]$ hardnesses, inapproximability

Reduction from INDEPENDENT SET:

Theorem

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

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction:

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction: given $k \leq |V|$ and $r \leq 1$:
Same reduction, we ask for a $(k+1)$-partition

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction: given $k \leq |V|$ and $r \leq 1$:
Same reduction, we ask for a $(k+1)$-partition

- if $\alpha(G) \geq k$ then $OPT(G') \leq k$

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction: given $k \leq |V|$ and $r \leq 1$:
Same reduction, we ask for a $(k+1)$-partition

- if $\alpha(G) \geq k$ then $OPT(G') \leq k$
- if $\alpha(G) \leq r.k$ then $OPT(G') \geq k + x$
  where x = minimum number of edges of any graph with k nodes that does not contain an independent set of size $r.k$

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction: given $k \leq |V|$ and $r \leq 1$:
Same reduction, we ask for a $(k+1)$-partition

- if $\alpha(G) \geq k$ then $OPT(G') \leq k$
- if $\alpha(G) \leq r.k$ then $OPT(G') \geq k + x$
  where $x =$ minimum number of edges of any graph with k nodes that does not contain an independent set of size $r.k$

Turan's theorem

Given G on n vertices and m edges : $\alpha(G) \geq \frac{n^2}{2m+n}$

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction: given $k \leq |V|$ and $r \leq 1$:
Same reduction, we ask for a $(k+1)$-partition

- if $\alpha(G) \geq k$ then $OPT(G') \leq k$
- if $\alpha(G) \leq r.k$ then $OPT(G') \geq k + x$
  where $x =$ minimum number of edges of any graph with k nodes that does not contain an independent set of size $r.k$

Turan's theorem

Given G on n vertices and m edges : $\alpha(G) \geq \frac{n^2}{2m+n}$

We have:

$$OPT(G') \geq \frac{1}{2r}k$$

Theorem

SUM-MAX GRAPH PARTITIONING cannot be approximated within $O(n^{1-\epsilon})$

Proof: gap preserving reduction: given $k \leq |V|$ and $r \leq 1$:
Same reduction, we ask for a $(k+1)$-partition

- if $\alpha(G) \geq k$ then $OPT(G') \leq k$
- if $\alpha(G) \leq r.k$ then $OPT(G') \geq k + x$
  where $x =$ minimum number of edges of any graph with k nodes that does not contain an independent set of size $r.k$

Turan's theorem

Given G on n vertices and m edges : $\alpha(G) \geq \frac{n^2}{2m+n}$

We have:

$$OPT(G') \geq \frac{1}{2r} k$$

$\Rightarrow$ gap preserved :
$O(n^{1-\epsilon})$ non approximable unless $\mathcal{P} = \mathcal{NP}$

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 ?

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 ?
- 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 ?
- restricted graph classes :
  - Polynomially solvable in interval graphs (for fixed k)
  - $\mathcal{NP}$-hard in split graphs

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 ?
- restricted graph classes :
  - Polynomially solvable in interval graphs (for fixed k)
  - $\mathcal{NP}$-hard in split graphs

  $\Rightarrow$ chordal graphs ?

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 ?
- restricted graph classes :
  - Polynomially solvable in interval graphs (for fixed k)
  - $\mathcal{NP}$-hard in split graphs

  $\Rightarrow$ chordal graphs ?
- links with graph homomorphisms, edge modification problems...

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 ?
- restricted graph classes :
  - Polynomially solvable in interval graphs (for fixed k)
  - $\mathcal{NP}$-hard in split graphs

  $\Rightarrow$ chordal graphs ?
- links with graph homomorphisms, edge modification problems...
- applications in software engineering:

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 ?
- restricted graph classes :
  - Polynomially solvable in interval graphs (for fixed k)
  - $\mathcal{NP}$-hard in split graphs

  $\Rightarrow$ chordal graphs ?
- links with graph homomorphisms, edge modification problems...
- applications in software engineering:
  adding/relaxing constraints ?

Thank you for your attention!