FPT algorithm for a generalized cut problem and some applications

EunJung Kim\textsuperscript{1} \quad Sang-Il Oum\textsuperscript{2} \\
Christophe Paul\textsuperscript{3} \quad Ignasi Sau\textsuperscript{3} \quad Dimitrios M. Thilikos\textsuperscript{3}

Séminaire AlGCo, January 2015

\textsuperscript{1} CNRS, LAMSADE, Paris (France)

\textsuperscript{2} KAIST, Daejeon (South Korea)

\textsuperscript{3} CNRS, LIRMM, Montpellier (France)
Outline of the talk

1. Introduction
2. Sketch of the FPT algorithm
3. Some applications
4. Conclusions
Next section is...

1 Introduction

2 Sketch of the FPT algorithm

3 Some applications

4 Conclusions
Some words on parameterized complexity

- **Idea:** given an NP-hard problem with input size $n$, fix one parameter $k$ of the input to see whether the problem gets more “tractable”.

**Example:** the size of a **Vertex Cover**.

- Given a (NP-hard) problem with input of size $n$ and a parameter $k$, a fixed-parameter tractable (FPT) algorithm runs in time

  $$f(k) \cdot n^{O(1)},$$

  for some function $f$.

**Examples:** $k$-**Vertex Cover**, $k$-**Longest Path**.
Many cut problems have been proved to be FPT

Cut problem: given a graph, find a minimum (vertex or edge) cutset whose removal makes the graph satisfy some separation property.
Many cut problems have been proved to be FPT

**Cut problem** given a graph, find a minimum (vertex or edge) cutset whose removal makes the graph satisfy some separation property.

- **Min Cut**: polynomial by classical max-flow min-cut theorem.
Many cut problems have been proved to be FPT

Cut problem: given a graph, find a minimum (vertex or edge) cutset whose removal makes the graph satisfy some separation property.

- **Min Cut**: polynomial by classical max-flow min-cut theorem.

- **Multiway Cut**: FPT by using important separators. [Marx '06]

- **Steiner Cut**: Improved FPT algorithm by using randomized contractions. [Chitnis, Cygan, Hajiaghayi, Pilipczuk '12]

- **Min Bisection**: Finally, FPT. [Cygan, Lokshtanov, Pilipczuk, Saurabh '13]
Many cut problems have been proved to be FPT

**Cut problem** given a graph, find a minimum (vertex or edge) cutset whose removal makes the graph satisfy some separation property.

- **Min Cut**: polynomial by classical max-flow min-cut theorem.
- **Multiway Cut**: FPT by using important separators. [Marx '06]
- **Multicut**: Finally, FPT. [Marx, Razgon + Bousquet, Daligault, Thomassé '06]
- **Steiner Cut**: Improved FPT algorithm by using randomized contractions. [Chitnis, Cygan, Hajiaghayi, Pilipczuk '12]

**Min Bisection**: Finally, FPT. [Cygan, Lokshtanov, Pilipczuk, Saurabh '13]
Many cut problems have been proved to be FPT

**Cut problem** given a graph, find a minimum (vertex or edge) cutset whose removal makes the graph satisfy some separation property.

- **Min Cut**: polynomial by classical max-flow min-cut theorem.
- **Multiway Cut**: FPT by using important separators. [Marx '06]
- **Multicut**: Finally, FPT. [Marx, Razgon + Bousquet, Daligault, Thomassé '06]
- **Steiner Cut**: Improved FPT algorithm by using randomized contractions. [Chitnis, Cygan, Hajiaghayi, Pilipczuk'12]
Many cut problems have been proved to be FPT

- **Cut problem**: given a graph, find a minimum (vertex or edge) cutset whose removal makes the graph satisfy some separation property.

- **Min Cut**: polynomial by classical max-flow min-cut theorem.

- **Multiway Cut**: FPT by using important separators. [Marx '06]

- **Multicut**: Finally, FPT. [Marx, Razgon + Bousquet, Daligault, Thomassé '06]

- **Steiner Cut**: Improved FPT algorithm by using randomized contractions. [Chitnis, Cygan, Hajiaghayi, Pilipczuk² '12]

- **Min Bisection**: Finally, FPT. [Cygan, Lokshtanov,Pilipczuk², Saurabh '13]
An \textit{r-allocation} of a set $S$ is an $r$-tuple $\mathcal{V} = (V_1, \ldots, V_r)$ of possibly empty sets that are pairwise disjoint and whose union is the set $S$.

Elements of $\mathcal{V}$: \textit{parts} of $\mathcal{V}$.

We denote by $\mathcal{V}^{(i)}$ the $i$-th part of $\mathcal{V}$, i.e., $\mathcal{V}^{(i)} = V_i$. 
An $r$-allocation of a set $S$ is an $r$-tuple $\mathcal{V} = (V_1, \ldots, V_r)$ of possibly empty sets that are pairwise disjoint and whose union is the set $S$.

Elements of $\mathcal{V}$: parts of $\mathcal{V}$.

We denote by $\mathcal{V}^{(i)}$ the $i$-th part of $\mathcal{V}$, i.e., $\mathcal{V}^{(i)} = V_i$.

Let $G = (V, E)$ be a graph and let $\mathcal{V}$ is an $r$-allocation of $V$:

$|\delta(\mathcal{V}^{(i)}, \mathcal{V}^{(j)})|$: #edges in $G$ with one endpoint in $\mathcal{V}^{(i)}$ and one in $\mathcal{V}^{(j)}$. 
Definition of the problem: **List Allocation**

**List Allocation**

**Input:** A tuple \( I = (G, r, \lambda, \alpha) \), where \( G \) is an \( n \)-vertex graph, \( r \in \mathbb{Z}_{\geq 1} \), \( \lambda : V(G) \rightarrow 2^r \), and \( \alpha : \binom{r}{2} \rightarrow \mathbb{Z}_{\geq 0} \).
Definition of the problem: \textbf{List Allocation}

\textbf{List Allocation}

\textbf{Input:} A tuple $l = (G, r, \lambda, \alpha)$, where $G$ is an $n$-vertex graph, $r \in \mathbb{Z}_{\geq 1}$, $\lambda : V(G) \rightarrow 2^r$, and $\alpha : \binom{2^r}{2} \rightarrow \mathbb{Z}_{\geq 0}$.

\textbf{Parameter:} $k = \sum \alpha$. 
Definition of the problem: **List Allocation**

**List Allocation**

**Input:** A tuple $I = (G, r, \lambda, \alpha)$, where $G$ is an $n$-vertex graph, $r \in \mathbb{Z}_{\geq 1}$, $\lambda : V(G) \rightarrow 2^r$, and $\alpha : \binom{r}{2} \rightarrow \mathbb{Z}_{\geq 0}$.

**Parameter:** $k = \sum \alpha$.

**Question:** Decide whether there exists an $r$-allocation $\mathcal{V}$ of $V(G)$ s.t.

- $\forall \{i, j\} \in \binom{r}{2}$, $|\delta(\mathcal{V}(i), \mathcal{V}(j))| = \alpha(i, j)$ and
- $\forall v \in V(G), \forall i \in [r]$, if $v \in \mathcal{V}(i)$ then $i \in \lambda(v)$.
Our main result

**Theorem**

*List Allocation* can be solved in time \(2^{O(k^2 \log k)} \cdot n^4 \cdot \log n\).

- *List Allocation* generalizes, in particular, the *Edge Multiway Cut-Uncut* problem.

- Our algorithm is strongly inspired by the *edge contraction* technique.

[Chitnis, Cygan, Hajiaghayi, Pilipczuk '12]
Introduction

2 Sketch of the FPT algorithm

3 Some applications

4 Conclusions
We use a series of FPT reductions:

Problem $A \xrightarrow{\text{FPT}}$ Problem $B$: If problem $B$ is FPT, then problem $A$ is FPT.
High-level ideas of the FPT algorithm

- We use a series of FPT reductions:
  \[ \text{Problem } A \xrightarrow{\text{FPT}} \text{Problem } B: \]  
  If problem \( B \) is FPT, then problem \( A \) is FPT.

- At some steps, we obtain instances whose size is bounded by some function \( f(k) \).

- Then we will use that the \textsc{List Allocation} problem is in XP:

**Lemma**

*There exists an algorithm that, given an instance \( I = (G, r, \lambda, \alpha) \) of \textsc{List Allocation}, computes all possible solutions in time \( n^{O(k)} \cdot r^{O(k+\ell)} \), where \( \ell \) is the number of connected components of \( G \).*
Some preliminaries

- Let $G$ be a connected graph. A partition $(V_1, V_2)$ of $V(G)$ is a $(q, k)$-separation if $|V_1|, |V_2| > q$, $|\delta(V_1, V_2)| \leq k$, and $G[V_1]$ and $G[V_2]$ are both connected.
Some preliminaries

- Let $G$ be a connected graph. A partition $(V_1, V_2)$ of $V(G)$ is a $(q, k)$-separation if $|V_1|, |V_2| > q$, $|\delta(V_1, V_2)| \leq k$, and $G[V_1]$ and $G[V_2]$ are both connected.

- A graph $G$ is $(q, k)$-connected if it does not contain any $(q, k)$-separation.
Some preliminaries

- Let $G$ be a connected graph. A partition $(V_1, V_2)$ of $V(G)$ is a $(q, k)$-separation if $|V_1|, |V_2| > q$, $|\delta(V_1, V_2)| \leq k$, and $G[V_1]$ and $G[V_2]$ are both connected.

- A graph $G$ is $(q, k)$-connected if it does not contain any $(q, k)$-separation.

Lemma (Chitnis, Cygan, Hajiaghayi, Pilipczuk'12)

There exists an algorithm that given a $n$-vertex connected graph $G$ and two integers $q, k$, either finds a $(q, k)$-separation, or reports that no such separation exists, in time $\min\{q, k\}^{O(\log(q+k))} n^3 \log n$. 
Series of FPT reductions

**List Allocation (LA)**
Series of FPT reductions

**List Allocation (LA)**

\[ \downarrow \text{FPT} \]

**Connected List Allocation (CLA)**

Same input + graph $G$ is connected and $r \leq 2k$
Series of FPT reductions

**List Allocation (LA)**

↓ FPT

**Connected List Allocation (CLA)**

↓ FPT

**Highly Connected List Allocation (HCLA)**

Same input + graph $G$ is $(f_1(k), k)$-connected, for $f_1(k) := 2^k \cdot (2k)^{2k}$
Series of FPT reductions

\textbf{List Allocation (LA)}
\[ \downarrow \text{FPT} \]
\textbf{Connected List Allocation (CLA)}
\[ \downarrow \text{FPT} \]
\textbf{Highly Connected List Allocation (HCLA)}

Same input + graph $G$ is $(f_1(k), k)$-connected, for $f_1(k) := 2^k \cdot (2k)^{2k}$

\textbf{Claim (Unique big part)}

For any solution $\mathcal{V}$ of HCLA there exists a unique index $j \in [r]$ such that

$$\sum_{i \in [r] \setminus j} |\mathcal{V}^{(i)}| \leq k \cdot f_1(k).$$

- Part $\mathcal{V}^{(j)}$ is called the \textbf{big part}.
- We say that $\mathcal{V}$ is $k \cdot f_1(k)$-bounded out of $j$. 
Reduction from CLA to HCLA: we shrink the graph

- We apply to $G$ the following recursive algorithm `shrink`, which receives a graph $G$ and a boundary set $B$ with $|B| \leq 2k$ (start with $B = \emptyset$):

  1. If $G$ has a $(f_1(k), k)$-separation $(V_1, V_2)$:
     - Without loss of generality, let $V_1$ be the part with the smallest number of boundary vertices, and let $B' = B$.
     - So $|B'| \leq 2k$.
     - Call recursively `shrink` with input $(G[V_1], B')$, and update the graph.

  2. Otherwise, we find a set of marginal vertices, and we identify them.

   Idea: We generate all possible behaviors of the boundary, and for each of them we compute a solution of HCLA, using our "black box".
Reduction from CLA to HCLA: we shrink the graph

- We apply to $G$ the following recursive algorithm $\text{shrink}$, which receives a graph $G$ and a boundary set $B$ with $|B| \leq 2k$ (start with $B = \emptyset$):
  1. If $G$ has a $(f_1(k), k)$-separation $(V_1, V_2)$:
     - W.l.o.g. let $V_1$ be the part with the smallest number of boundary vertices, and let $B'$ be the new boundary: so $|B'| \leq 2k$.
     - Call recursively $\text{shrink}$ with input $(G[V_1], B')$, and update the graph.
Reduction from CLA to HCLA: we shrink the graph

- We apply to $G$ the following recursive algorithm $\text{shrink}$, which receives a graph $G$ and a boundary set $B$ with $|B| \leq 2k$ (start with $B = \emptyset$):

  1. If $G$ has a $(f_1(k), k)$-separation ($V_1, V_2$):
     - W.l.o.g. let $V_1$ be the part with the smallest number of boundary vertices, and let $B'$ be the new boundary: so $|B'| \leq 2k$.
     - Call recursively $\text{shrink}$ with input $(G[V_1], B')$, and update the graph.

  2. Otherwise, we find a set of marginal vertices, and we identify them.

         **Idea** We generate all possible behaviors of the boundary, and for each of them we compute a solution of HCLA, using our “black box”.

![Diagram of a graph with parts and vertices labeled $V_1$, $V_2$, $B$, and $f_1(k)$]
Reduction from CLA to HCLA: we shrink the graph

- We apply to $G$ the following recursive algorithm \texttt{shrink}, which receives a graph $G$ and a boundary set $B$ with $|B| \leq 2k$ (start with $B = \emptyset$):
  
  1. If $G$ has a $(f_1(k), k)$-separation $(V_1, V_2)$:
     - W.l.o.g. let $V_1$ be the part with the smallest number of boundary vertices, and let $B'$ be the new boundary: so $|B'| \leq 2k$.
     - Call recursively \texttt{shrink} with input $(G[V_1], B')$, and update the graph.
  
  2. Otherwise, we find a set of marginal vertices, and we identify them.

  **Idea** By the high connectivity (\text{Claim}), each such solution has a unique big part $\mathcal{V}^{(j)}$: these are the marginal vertices for this behavior.
Reduction from CLA to HCLA: we shrink the graph

- We apply to $G$ the following recursive algorithm `shrink`, which receives a graph $G$ and a boundary set $B$ with $|B| \leq 2k$ (start with $B = \emptyset$):

  1. If $G$ has a $(f_1(k), k)$-separation $(V_1, V_2)$:
     - W.l.o.g. let $V_1$ be the part with the smallest number of boundary vertices, and let $B'$ be the new boundary: so $|B'| \leq 2k$.
     - Call recursively `shrink` with input $(G[V_1], B')$, and update the graph.

  2. Otherwise, we find a set of marginal vertices, and we identify them.

     **Idea** If the graph is big enough, there are vertices that are marginal for all behaviors $\Rightarrow$ they can be safely identified. Return the graph.
Reduction from CLA to HCLA: we shrink the graph

- We apply to $G$ the following recursive algorithm $\text{shrink}$, which receives a graph $G$ and a boundary set $B$ with $|B| \leq 2k$ (start with $B = \emptyset$):
  1. If $G$ has a $(f_1(k), k)$-separation $(V_1, V_2)$:
     - W.l.o.g. let $V_1$ be the part with the smallest number of boundary vertices, and let $B'$ be the new boundary: so $|B'| \leq 2k$.
     - Call recursively $\text{shrink}$ with input $(G[V_1], B')$, and update the graph.
  2. Otherwise, we find a set of marginal vertices, and we identify them.

   **Idea** If the graph is big enough, there are vertices that are marginal for all behaviors $\Rightarrow$ they can be safely identified. Return the graph.

**Lemma**

The above algorithm returns in FPT time an equivalent instance of CLA of size at most $f_2(k) := k \cdot (f_1(k))^2 + 2k + 2$. (Then we apply the XP algorithm.)
Series of FPT reductions

List Allocation (LA) ↓ FPT

Connected List Allocation (CLA) ↓ FPT

Highly Connected List Allocation (HCLA)
Series of FPT reductions

**List Allocation (LA)**

\[ \downarrow \text{FPT} \]

**Connected List Allocation (CLA)**

\[ \downarrow \text{FPT} \]

**Highly Connected List Allocation (HCLA)**

\[ \downarrow \text{FPT} \]

**Split Highly Connected List Allocation (SHCLA)**

Same input + set \( S \subseteq V(G) \) and a solution \( \mathcal{V} \) additionally needs to satisfy that there exists some \( j \in [r] \) such that

A. \( \mathcal{V} \) is \( k \cdot f_1(k) \)-bounded out of \( j \) and

B. \( \partial \mathcal{V}(j) \subseteq S \subseteq \mathcal{V}(j) \).
Crucial ingredient: Splitter Lemma

- Splitters were first introduced by [Naor, Schulman, Srinivasan '95]
- We use the following deterministic version:

**Lemma (Chitnis, Cygan, Hajiaghayi, Pilipczuk '12)**

> There exists an algorithm that given a set $U$ of size $n$ and two integers $a, b \in [0, n]$, outputs a set $F \subseteq 2^U$ where $|F| = \min\{a, b\}^O(\log(a+b)) \cdot \log n$ such that for every two sets $A, B \subseteq U$, where $A \cap B = \emptyset$, $|A| \leq a$, $|B| \leq b$, there exists a set $S \in F$ where $A \subseteq S$ and $B \cap S = \emptyset$, in $\min\{a, b\}^O(\log(a+b)) \cdot n \log n$ steps.
We use the **Splitter Lemma** with universe $U = V(G)$, $a = k$, and $b = k \cdot f_1(k)$, obtaining a family $\mathcal{F}$ of subsets of $V(G)$. 

Idea

We want a set $S \subseteq V(G)$ that "splits" these two sets: $A = \partial V_G(j)$ and $B = \bigcup_{i \in [r]} \{j\} V_G(i)$.

For some $j \in [r]$: $|A| \leq k$ and $|B| \leq k \cdot f_1(k)$ (by the Claim).

It holds that $I$ is a **Yes**-instance of HCLA if and only if for some $S \in F$, $(I, S)$ is a **Yes**-instance of SHCLA.
Reduction from HCLA to SHCLA: we use splitters

- We use the **Splitter Lemma** with universe $U = V(G)$, $a = k$, and $b = k \cdot f_1(k)$, obtaining a family $\mathcal{F}$ of subsets of $V(G)$.
- **Idea** We want a set $S \subseteq V(G)$ that “splits” these two sets:

  $$A = \partial V^{(j)} \quad \text{and} \quad B = \bigcup_{i \in [r]\setminus\{j\}} V^{(i)}.$$

For some $j \in [r]$: $|A| \leq k$ and $|B| \leq k \cdot f_1(k)$ (by the **Claim**).
We use the **Splitter Lemma** with universe $U = V(G)$, $a = k$, and $b = k \cdot f_1(k)$, obtaining a family $\mathcal{F}$ of subsets of $V(G)$.

**Idea** We want a set $S \subseteq V(G)$ that “splits” these two sets:

$$A = \partial V^{(j)} \text{ and } B = \bigcup_{i \in [r]\setminus\{j\}} V^{(i)}.$$ 

For some $j \in [r]$: $|A| \leq k$ and $|B| \leq k \cdot f_1(k)$ (by the Claim).

It holds that $I$ is a **Yes**-instance of **HCLA** if and only if for some $S \in \mathcal{F}$, $(I, S)$ is a **Yes**-instance of **SHCLA**.
An algorithm to solve SHCLA

- Try all $j \in [r]$ so that $V^{(j)}$ is the big part: assume $\partial V^{(j)} \subseteq S \subseteq V^{(j)}$. 

Lemma

The SHCLA problem can be solved in time $2^{O(k^2 \cdot \log k)} \cdot n$. 
An algorithm to solve SHCLA

- Try all $j \in [r]$ so that $\mathcal{V}(j)$ is the big part: assume $\partial \mathcal{V}(j) \subseteq S \subseteq \mathcal{V}(j)$.
- Partition the connected components of $G \setminus S$ into 3 sets:
  - $\mathcal{W}$: those that are small ($\leq f_1(k)$) and that can go entirely in $\mathcal{V}(j)$.
  - $\mathcal{Z}$: those that are big ($> f_1(k)$) and that can go entirely in $\mathcal{V}(j)$.
  - $\mathcal{Y}$: those that cannot go entirely in $\mathcal{V}(j)$. 

Lemma

The SHCLA problem can be solved in time $2^{O(k^2 \log k)} \cdot n$. 

\[ \text{Lemma} \]

The SHCLA problem can be solved in time $2^{O(k^2 \log k)} \cdot n$. 

\[ \text{Lemma} \]

The SHCLA problem can be solved in time $2^{O(k^2 \log k)} \cdot n$. 

\[ \text{Lemma} \]
An algorithm to solve SHCLA

- Try all $j \in [r]$ so that $\mathcal{V}(j)$ is the big part: assume $\partial \mathcal{V}(j) \subseteq S \subseteq \mathcal{V}(j)$.
- Partition the connected components of $G \setminus S$ into 3 sets:
  - $\mathcal{W}$: those that are small ($\leq f_1(k)$) and that can go entirely in $\mathcal{V}(j)$.
  - $\mathcal{Z}$: those that are big ($> f_1(k)$) and that can go entirely in $\mathcal{V}(j)$.
  - $\mathcal{Y}$: those that cannot go entirely in $\mathcal{V}(j)$.

**Lemma**

The SHCLA problem can be solved in time $2^{O(k^2 \cdot \log k)} \cdot n$. 

17/25
Piecing everything together

**List Allocation (LA)**

↓ FPT reduction

**Connected List Allocation (CLA)**

↓ FPT reduction

**Highly Connected List Allocation (HCLA)**

↓ FPT reduction

**Split Highly Connected List Allocation (SHCLA)**

↓ FPT algorithm to solve SHCLA

**Theorem**

**List Allocation** can be solved in time $2^{O(k^2 \log k)} \cdot n^4 \cdot \log n$. 
Generalization of **Digraph Homomorphism**

**Arc-Bounded List Digraph Homomorphism**

**Input:** Two digraphs $G$ and $H$, a list $\lambda : V(G) \to 2^{V(H)}$ of allowed images for every vertex in $G$, and a function $\alpha$ prescribing the number of arcs in $G$ mapped to each arc of $H$.

Parameter: $k = \sum \alpha$.

**Question:** Decide whether there exists a homomorphism from $G$ to $H$ respecting the constraints imposed by $\lambda$ and $\alpha$.

It generalizes several homomorphism problems. 

[D ´ıaz, Serna, Thilikos '08]

**Corollary**
The **Arc-Bounded List Digraph Homomorphism** problem is **FPT**.
Generalization of **Digraph Homomorphism**

**Arc-Bounded List Digraph Homomorphism**

*Input*: Two digraphs $G$ and $H$, a list $\lambda : V(G) \rightarrow 2^{V(H)}$ of allowed images for every vertex in $G$, and a function $\alpha$ prescribing the number of arcs in $G$ mapped to each arc of $H$.

*Parameter*: $k = \sum \alpha$. 
Generalization of **Digraph Homomorphism**

**Arc-Bounded List Digraph Homomorphism**

**Input:** Two digraphs $G$ and $H$, a list $\lambda : V(G) \rightarrow 2^{V(H)}$ of allowed images for every vertex in $G$, and a function $\alpha$ prescribing the number of arcs in $G$ mapped to each arc of $H$.

**Parameter:** $k = \sum \alpha$.

**Question:** Decide whether there exists a homomorphism from $G$ to $H$ respecting the constraints imposed by $\lambda$ and $\alpha$.

- It generalizes several homomorphism problems.  

[Díaz, Serna, Thilikos ’08]
**Generalization of Digraph Homomorphism**

**Arc-Bounded List Digraph Homomorphism**

**Input:** Two digraphs $G$ and $H$, a list $\lambda : V(G) \to 2^{V(H)}$ of allowed images for every vertex in $G$, and a function $\alpha$ prescribing the number of arcs in $G$ mapped to each arc of $H$.

**Parameter:** \( k = \sum \alpha \).

**Question:** Decide whether there exists a homomorphism from $G$ to $H$ respecting the constraints imposed by $\lambda$ and $\alpha$.

- It generalizes several homomorphism problems. [Díaz, Serna, Thilikos '08]

**Corollary**

*The Arc-Bounded List Digraph Homomorphism problem is FPT.*
Graph partitioning problem

**Min-Max Graph Partitioning**

**Input**: An undirected graph $G$, $w, r \in \mathbb{Z}_{\geq 0}$, and $T \subseteq V(G)$ with $|T| = r$. 
**Min-Max Graph Partitioning**

**Input:** An undirected graph $G$, $w, r \in \mathbb{Z}_{\geq 0}$, and $T \subseteq V(G)$ with $|T| = r$.

**Parameter:** $k = w \cdot r$. 

**Question:** Decide whether there exists a partition $\{P_1, \ldots, P_r\}$ of $V(G)$ such that $\max_{i \in [r]} |\delta(P_i, V(G) \setminus P_i)| \leq w$ and for every $i \in [r], |P_i \cap T| = 1$.

**Important in approximation.**

[Bansal, Feige, Krauthgamer, Makarychev, Nagarajan, Naor, Schwartz'11]

The "Min-Sum" version is exactly the Multiway Cut problem.

[Marx '06]

**Corollary** The Min-Max Graph Partitioning problem is FPT.
Graph partitioning problem

**Min-Max Graph Partitioning**

**Input:** An undirected graph $G$, $w, r \in \mathbb{Z}_{\geq 0}$, and $T \subseteq V(G)$ with $|T| = r$.

**Parameter:** $k = w \cdot r$.

**Question:** Decide whether there exists a partition $\{P_1, \ldots, P_r\}$ of $V(G)$ such that $\max_{i \in [r]} |\delta(P_i, V(G) \setminus P_i)| \leq w$ and for every $i \in [r]$, $|P_i \cap T| = 1$.

- Important in approximation. [Bansal, Feige, Krauthgamer, Makarychev, Nagarajan, Naor, Schwartz’11]
- The “Min-Sum” version is exactly the Multiway Cut problem. [Marx ’06]
**Graph partitioning problem**

**Min-Max Graph Partitioning**

**Input:** An undirected graph $G$, $w, r \in \mathbb{Z}_{\geq 0}$, and $T \subseteq V(G)$ with $|T| = r$.

**Parameter:** $k = w \cdot r$.

**Question:** Decide whether there exists a partition $\{P_1, \ldots, P_r\}$ of $V(G)$ s.t. $\max_{i \in [r]} |\delta(P_i, V(G) \setminus P_i)| \leq w$ and for every $i \in [r]$, $|P_i \cap T| = 1$.

- Important in approximation. [Bansal, Feige, Krauthgamer, Makarychev, Nagarajan, Naor, Schwartz'11]
- The “Min-Sum” version is exactly the Multiway Cut problem. [Marx '06]

**Corollary**

The Min-Max Graph Partitioning problem is FPT.
2-approximation for **Tree-cut width**

- **Tree-cut width** is a graph invariant fundamental in the structure of graphs not admitting a fixed graph as an immersion. [Wollan '14]

- **Tree-cut decompositions** are a variation of tree decompositions based on edge cuts instead of vertex cuts.

- Tree-cut width also has algorithmic applications. [Ganian, Kim, Szeider'14]
2-approximation for **Tree-cut width**

- **Tree-cut width** is a graph invariant fundamental in the structure of graphs not admitting a fixed graph as an **immersion**. [Wollan '14]

- **Tree-cut decompositions** are a variation of tree decompositions based on edge cuts instead of vertex cuts.

- Tree-cut width also has **algorithmic applications**. [Ganian, Kim, Szeider'14]

- We prove that following result:

**Corollary**

*There exists an algorithm that, given a graph $G$ and a $k \in \mathbb{Z}_{\geq 0}$, in time $2^{O(k^2 \cdot \log k)} \cdot n^5 \cdot \log n$ either outputs a tree-cut decomposition of $G$ with width at most $2k$, or correctly reports that the tree-cut width of $G$ is strictly larger than $k$.***
1 Introduction

2 Sketch of the FPT algorithm

3 Some applications

4 Conclusions
Conclusions and further research

Some open problems:

- Improve the running time of our algorithms.
Some open problems:

- Improve the running time of our algorithms.
- Can we find more applications of List Allocation?
Conclusions and further research

Some open problems:

- Improve the running time of our algorithms.
- Can we find more applications of List Allocation?
- Find an explicit (exact) FPT algorithm for tree-cut width.
Conclusions and further research

Some open problems:

- Improve the running time of our algorithms.
- Can we find more applications of List Allocation?
- Find an explicit (exact) FPT algorithm for tree-cut width.
- Recent work on finding \((q, k)\)-separations: \cite{Montejano, S. '15}
  - FPT when parameterized by both \(q\) and \(k\).
  - \(W[1]\)-hard when parameterized by \(q\).
  - No polynomial kernel when parameterized by \(k\).
Conclusions and further research

Some open problems:

- Improve the running time of our algorithms.

- Can we find more applications of `List Allocation`?

- Find an explicit (exact) FPT algorithm for `tree-cut width`.

- Recent work on finding $(q, k)$-separations:
  - FPT when parameterized by both $q$ and $k$.
  - W[1]-hard when parameterized by $q$.
  - No polynomial kernel when parameterized by $k$.
  - ★ FPT when parameterized by $k$?

[Montejano, S. '15]
Gràcies!