FPT algorithm for a generalized cut problem and some applications

EunJung Kim[1] Sang-Il Oum[2]
Christophe Paul[3] Ignasi Sau[3] Dimitrios M. Thilikos[3]

Séminaire AlGCo, January 2015

[1] CNRS, LAMSADE, Paris (France)

[2] KAIST, Daejeon (South Korea)

[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...

Some words on parameterized complexity

- $\boxed{\text{Idea}}$ 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

$$f(k) \cdot n^{O(1)}, \quad \text{for some function } f.$$

  Examples: k-Vertex Cover, k-Longest Path.

Many cut problems have been proved to be FPT

$\boxed{\text{Cut problem}}$ 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

$\boxed{\text{Cut problem}}$ 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]

Many cut problems have been proved to be FPT

$\boxed{\text{Cut problem}}$ 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]

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^2 '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, Pilipczuk^2 '12]
- MIN BISECTION: Finally, FPT. [Cygan, Lokshtanov,Pilipczuk^2, Saurabh '13]

Definition of the problem – preliminaries

- 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., $\mathcal{V}^{(i)} = V_i$.

Definition of the problem – preliminaries

- 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 $\mathcal{V}^{(j)}$.

List Allocation

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

List Allocation

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

Parameter: $k = \sum \alpha$.

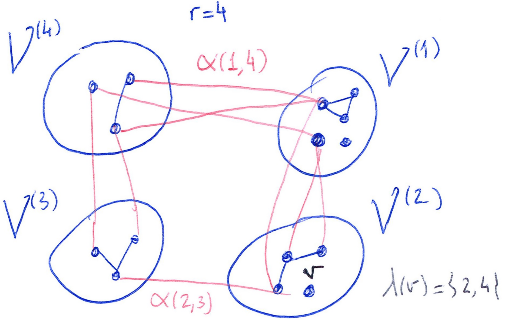
List Allocation

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

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^2 '12]

Next section is...

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

High-level ideas of the FPT algorithm

- 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:

Problem A $\xrightarrow{\text{FPT}}$ Problem B: If problem B is FPT, then problem A is FPT.

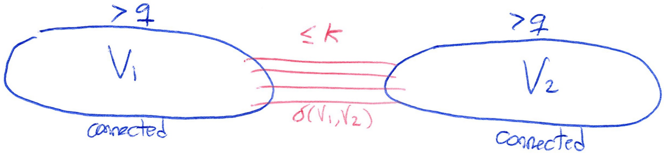
- At some steps, we obtain instances whose size is bounded by some function $f(k)$.
- Then we will use that the List Allocation problem is in XP:

Lemma

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

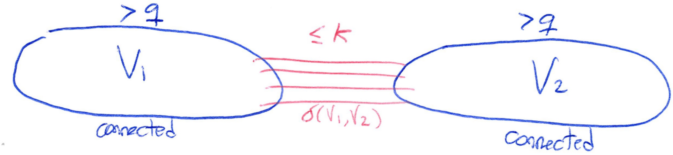
Some preliminaries

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



Some preliminaries

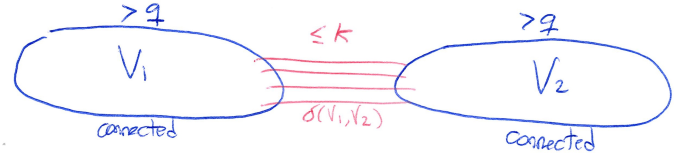
- Let G be a connected graph. A partition (V_1, V_2) of $V(G)$ is a (q, k)-separation if $|V_1|, |V_2| > q$, $|\delta(V_1, V_2)| \leqslant k$, and $G[V_1]$ and $G[V_2]$ are both connected.



- 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 $|V_1|, |V_2| > q$, $|\delta(V_1, V_2)| \leqslant k$, 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^2 '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

$\boxed{\text{List Allocation (LA)}}$

Series of FPT reductions

$\boxed{\text{List Allocation (LA)}}$

$\downarrow$ FPT

$\boxed{\text{Connected List Allocation (CLA)}}$

Same input + graph G is connected and $r \leqslant 2k$

Series of FPT reductions

$\boxed{\textsc{List Allocation (LA)}}$

$\downarrow$ FPT

$\boxed{\textsc{Connected List Allocation (CLA)}}$

$\downarrow$ FPT

$\boxed{\textsc{Highly Connected List Allocation (HCLA)}}$

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

Series of FPT reductions

LIST ALLOCATION (LA)

$\downarrow$ FPT

CONNECTED LIST ALLOCATION (CLA)

$\downarrow$ FPT

HIGHLY CONNECTED LIST ALLOCATION (HCLA)

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

Claim (Unique big part)

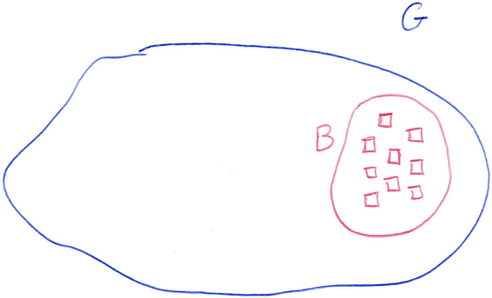
For any solution $\mathcal{V}$ of HCLA there exists a unique index $j \in [r]$ such that

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

- Part $\mathcal{V}^{(j)}$ is called the big part.
- We say that $\mathcal{V}$ is $k \cdot f_1(k)$-bounded out of j.

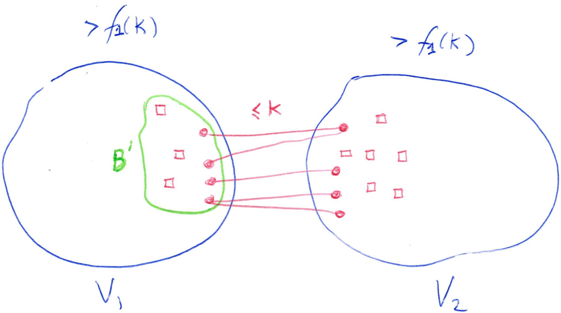
Reduction from CLA to HCLA: we shrink the graph

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



Reduction from CLA to HCLA: we shrink the graph

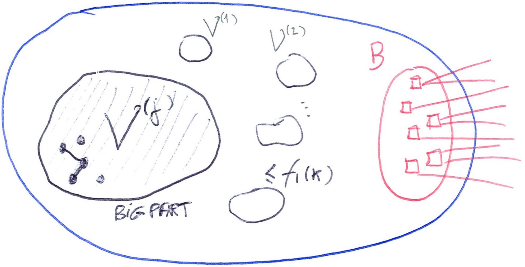
- We apply to G the following recursive algorithm `shrink`, which receives a graph G and a boundary set B with $|B| \leqslant 2k$ (start with $B = \emptyset$):
  1. If G has a $(f_1(k), k)$-separation (V_1, V_2):
     - W.l.o.g. let V_1 be the part with the smallest number of boundary vertices, and let B' be the new boundary: so $|B'| \leqslant 2k$.
     - Call recursively `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 `shrink`, which receives a graph G and a boundary set B with $|B| \leqslant 2k$ (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 $|B'| \leqslant 2k$.
     - 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.

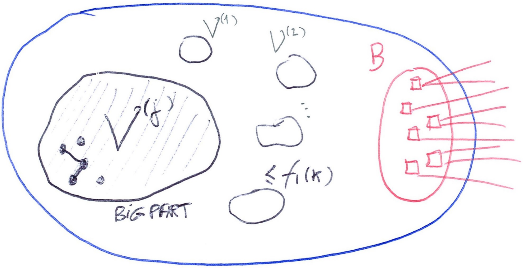
  Idea We generate all possible behaviors of the boundary, and for each of them we compute a solution of HCLA, using our "black box".



Reduction from CLA to HCLA: we shrink the graph

- We apply to G the following recursive algorithm `shrink`, which receives a graph G and a boundary set B with $|B| \leqslant 2k$ (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 $|B'| \leqslant 2k$.
     - 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.

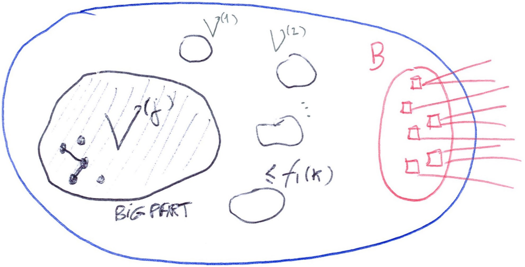
  $\boxed{\text{Idea}}$ By the high connectivity (Claim), each such solution has a unique big part $\mathcal{V}^{(j)}$: these are the marginal vertices for this behavior.



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 $|B| \leqslant 2k$ (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 $|B'| \leqslant 2k$.
     - 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 `shrink`, which receives a graph G and a boundary set B with $|B| \leqslant 2k$ (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 $|B'| \leqslant 2k$.
     - 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.

     $\boxed{\text{Idea}}$ 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

$\boxed{\text{LIST ALLOCATION (LA)}}$

$\downarrow$ FPT

$\boxed{\text{CONNECTED LIST ALLOCATION (CLA)}}$

$\downarrow$ FPT

$\boxed{\text{HIGHLY CONNECTED LIST ALLOCATION (HCLA)}}$

Series of FPT reductions

List Allocation (LA)

$\downarrow$ FPT

Connected List Allocation (CLA)

$\downarrow$ FPT

Highly Connected List Allocation (HCLA)

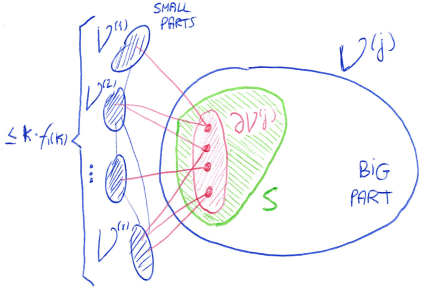
$\downarrow$ FPT

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. $\partial\mathcal{V}^{(j)} \subseteq S \subseteq \mathcal{V}^{(j)}$.

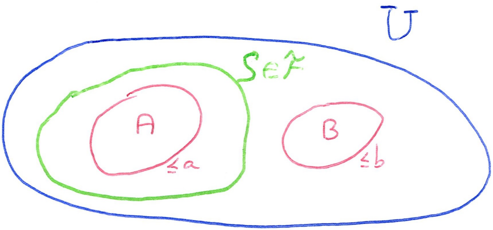


Crucial ingredient: Splitter Lemma

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

Lemma (Chitnis, Cygan, Hajiaghayi, Pilipczuk^2 '12)

There exists an algorithm that given a set U of size n and two integers $a, b \in [0, n]$, outputs a set $\mathcal{F} \subseteq 2^U$ where $|\mathcal{F}| = \min\{a, b\}^{O(\log(a+b))} \cdot \log n$ such that for every two sets $A, B \subseteq U$, where $A \cap B = \emptyset$, $|A| \leqslant a$, $|B| \leqslant b$, there exists a set $S \in \mathcal{F}$ where $A \subseteq S$ and $B \cap S = \emptyset$, in $\min\{a, b\}^{O(\log(a+b))} \cdot n \log n$ steps.



Reduction from HCLA to SHCLA: we use splitters

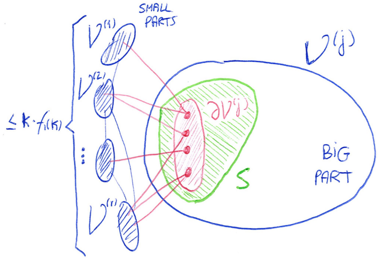
- We use the Splitter Lemma with universe $U = V(G)$, $a = k$, and $b = k \cdot f_1(k)$, obtaining a family $\mathcal{F}$ of subsets of $V(G)$.

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 $V(G)$.
- $\boxed{\text{Idea}}$ We want a set $S \subseteq V(G)$ that "splits" these two sets:

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

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

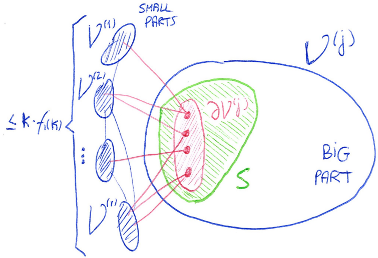


Reduction from HCLA to SHCLA: we use splitters

- 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 $V(G)$.
- $\boxed{\text{Idea}}$ We want a set $S \subseteq V(G)$ that "splits" these two sets:

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

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



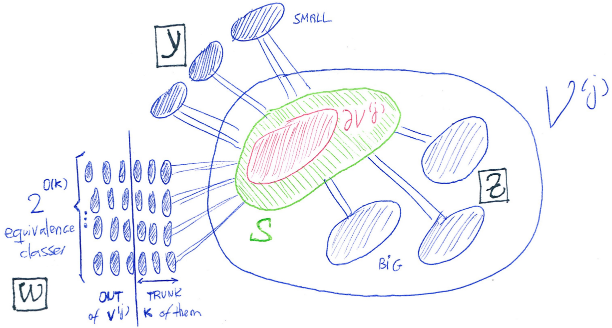
- It holds that I is a YES-instance of HCLA if and only if for some $S \in \mathcal{F}$, (I, S) is a YES-instance of SHCLA.

An algorithm to solve SHCLA

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

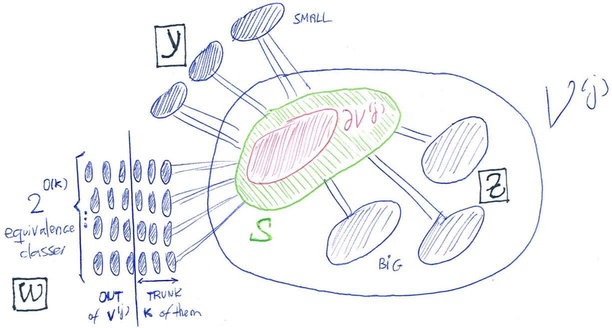
An algorithm to solve SHCLA

- 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 ($\leqslant f_1(k)$) 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 $\mathcal{V}^{(j)}$.



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 ($\leqslant f_1(k)$) 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 $2^{O(k^2 \cdot \log k)} \cdot n$.

Piecing everything together

List Allocation (LA)

$\downarrow$ FPT reduction

Connected List Allocation (CLA)

$\downarrow$ FPT reduction

Highly Connected List Allocation (HCLA)

$\downarrow$ FPT reduction

Split Highly Connected List Allocation (SHCLA)

$\downarrow$ FPT algorithm to solve SHCLA

Theorem

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

Next section is...

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 α prescribing the number of arcs in G mapped to each arc of H.

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 α prescribing the number of arcs in G mapped to each arc of H.

Parameter: $k = \sum \alpha$.

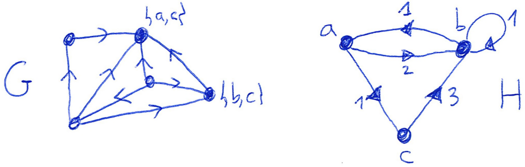
Generalization of DIGRAPH HOMOMORPHISM

ARC-BOUNDED LIST DIGRAPH HOMOMORPHISM

Input: Two digraphs G and H, a list $\lambda : V(G) \to 2^{V(H)}$ of allowed images for every vertex in G, and a function α 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 λ and α.



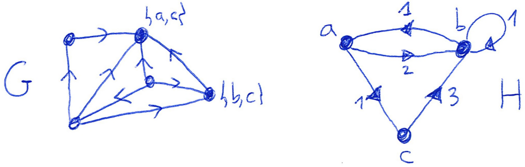
- It generalizes several homomorphism problems. [Díaz, Serna, Thilikos '08]

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 α 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 λ and α.



- 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, $w, r \in \mathbb{Z}_{\geqslant 0}$, and $T \subseteq V(G)$ with $|T| = r$.

Graph partitioning problem

MIN-MAX GRAPH PARTITIONING

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

Parameter: $k = w \cdot r$.

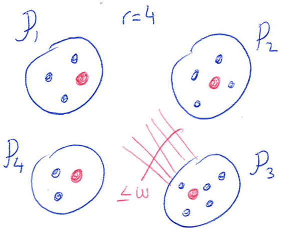
Graph partitioning problem

Min-Max Graph Partitioning

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

Parameter: $k = w \cdot r$.

Question: Decide whether there exists a partition $\{\mathcal{P}_1, \ldots, \mathcal{P}_r\}$ of $V(G)$ s.t. $\max_{i \in [r]} |\delta(\mathcal{P}_i, V(G) \setminus \mathcal{P}_i)| \leqslant w$ and for every $i \in [r]$, $|\mathcal{P}_i \cap T| = 1$.



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

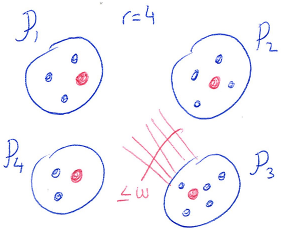
Graph partitioning problem

MIN-MAX GRAPH PARTITIONING

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

Parameter: $k = w \cdot r$.

Question: Decide whether there exists a partition $\{\mathcal{P}_1, \ldots, \mathcal{P}_r\}$ of $V(G)$ s.t. $\max_{i \in [r]} |\delta(\mathcal{P}_i, V(G) \setminus \mathcal{P}_i)| \leqslant w$ and for every $i \in [r]$, $|\mathcal{P}_i \cap T| = 1$.



- 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 $k \in \mathbb{Z}_{\geqslant 0}$, 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.

Next section is...

Conclusions and further research

Some open problems:

- Improve the running time of our algorithms.

Conclusions and further research

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: [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: [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.
  - ★ FPT when parameterized by k?

Gràcies!