Randomized techniques for parameterized algorithms

Dániel Marx[1]

[1]Computer and Automation Research Institute,
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

IPEC 2012
September 13, 2012
Ljubljana, Slovenia

Why randomized?

- A guaranteed error probability of 10^{-100} is as good as a deterministic algorithm.
  (Probability of hardware failure is larger!)
- Randomized algorithms can be more efficient and/or conceptually simpler.
- Can be the first step towards a deterministic algorithm.

Polynomial time vs. FPT

FPT

A parameterized problem is fixed-parameter tractable if it can be solved in time $f(k) \cdot n^{O(1)}$ for some computable function f.

Polynomial time vs. FPT

FPT

A parameterized problem is fixed-parameter tractable if it can be solved in time $f(k) \cdot n^{O(1)}$ for some computable function f.

Polynomial-time randomized algorithms

- Randomized selection to pick a typical, unproblematic, average element/subset.
- Error probability is constant or at most polynomially small.

Randomized FPT algorithms

- Randomized selection to satisfy a bounded number of (unknown) constraints.
- Error probability might be exponentially small.

Randomization

There are two main ways randomization appears:

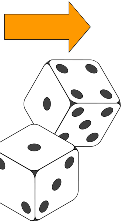
- Algebraic techniques (Schwartz-Zippel Lemma)
  See Andreas Björklund's talk, Friday 13:30.

- Combinatorial techniques.
  This talk.

Randomization as reduction

Problem A
(what we want to solve)

Randomized magic



Problem B
(what we can solve)

Color Coding

k-Path

Input: A graph G, integer k.
Find: A simple path of length k.

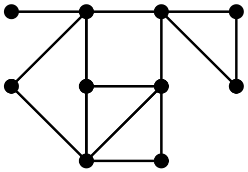
Note: The problem is clearly NP-hard, as it contains the Hamiltonian Path problem.

Theorem [Alon, Yuster, Zwick 1994]

k-Path can be solved in time $2^{O(k)} \cdot n^{O(1)}$.

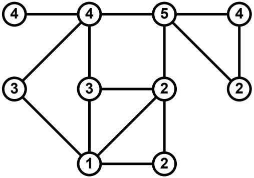
Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



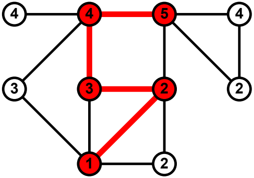
Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- Check if there is a path colored $1 - 2 - \cdots - k$; output "YES" or "NO".
  - If there is no k-path: no path colored $1 - 2 - \cdots - k$ exists $\Rightarrow$ "NO".
  - If there is a k-path: the probability that such a path is colored $1 - 2 - \cdots - k$ is k^{-k} thus the algorithm outputs "YES" with at least that probability.

Error probability

Useful fact

If the probability of success is at least p, then the probability that the algorithm **does not** say "YES" after $1/p$ repetitions is at most

$$(1-p)^{1/p} < \left(e^{-p}\right)^{1/p} = 1/e \approx 0.38$$

Error probability

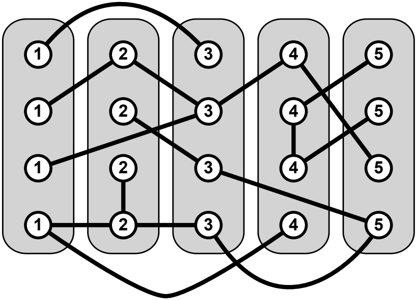
Useful fact

If the probability of success is at least p, then the probability that the algorithm **does not** say "YES" after $1/p$ repetitions is at most

$$(1-p)^{1/p} < \left(e^{-p}\right)^{1/p} = 1/e \approx 0.38$$

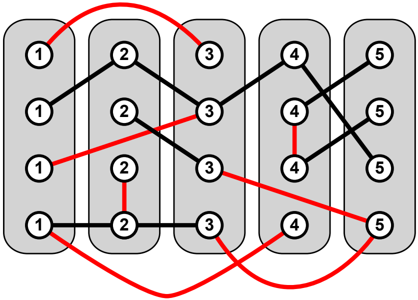
- Thus if $p > k^{-k}$, then error probability is at most $1/e$ after k^k repetitions.
- Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- For example, by trying $100 \cdot k^k$ random colorings, the probability of a wrong answer is at most $1/e^{100}$.

Finding a path colored $1 - 2 - \cdots - k$



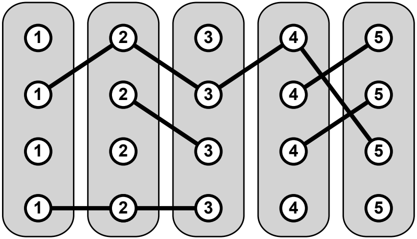
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$



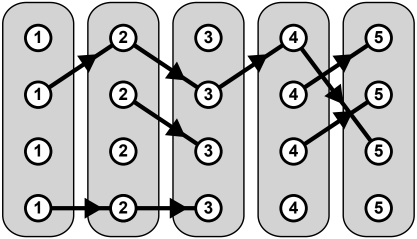
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$



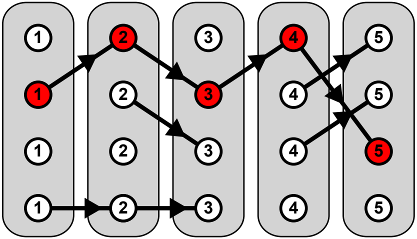
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$



- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Finding a path colored $1 - 2 - \cdots - k$

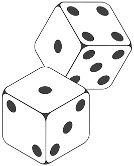


- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Color Coding

k-**PATH**

Color Coding
success probability:
k^{-k}

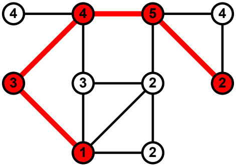


Finding a
$1 - 2 - \cdots - k$
colored path

polynomial-time
solvable

Improved Color Coding

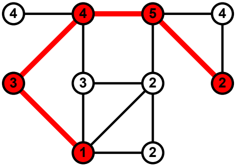
- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- Check if there is a colorful path where each color appears exactly once on the vertices; output "YES" or "NO".

Improved Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



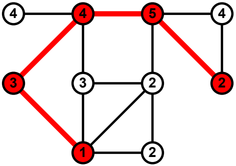
- Check if there is a colorful path where each color appears exactly once on the vertices; output "YES" or "NO".
  - If there is no k-path: no colorful path exists $\Rightarrow$ "NO".
  - If there is a k-path: the probability that it is colorful is

$$\frac{k!}{k^k} > \frac{\left(\frac{k}{e}\right)^k}{k^k} = e^{-k},$$

thus the algorithm outputs "YES" with at least that probability.

Improved Color Coding

- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- Repeating the algorithm $100e^k$ times decreases the error probability to e^{-100}.

How to find a colorful path?

- Try all permutations ($k! \cdot n^{O(1)}$ time)
- Dynamic programming ($2^k \cdot n^{O(1)}$ time)

Finding a colorful path

Subproblems:

We introduce $2^k \cdot |V(G)|$ Boolean variables:

$$x(v, C) = \text{TRUE for some } v \in V(G) \text{ and } C \subseteq [k]$$

$$\Updownarrow$$

There is a P path ending at v such that each color in C appears on P exactly once and no other color appears.

Answer:

There is a colorful path $\iff$ $x(v, [k]) = \text{TRUE}$ for some vertex v.

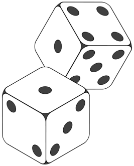
Initialization & Recurrence:

Exercise.

Improved Color Coding

k-**PATH**

Color Coding
success probability:
e^{-k}



Finding a colorful path

Solvable in time
$2^k \cdot n^{O(1)}$

Derandomization

Definition

A family $\mathcal{H}$ of functions $[n] \to [k]$ is a k-**perfect** family of hash functions if for every $S \subseteq [n]$ with $|S| = k$, there is an $h \in \mathcal{H}$ such that $h(x) \neq h(y)$ for any $x, y \in S$, $x \neq y$.

Theorem

There is a k-perfect family of functions $[n] \to [k]$ having size $2^{O(k)} \log n$ (and can be constructed in time polynomial in the size of the family).

Derandomization

Definition

A family $\mathcal{H}$ of functions $[n] \to [k]$ is a k-**perfect** family of hash functions if for every $S \subseteq [n]$ with $|S| = k$, there is an $h \in \mathcal{H}$ such that $h(x) \neq h(y)$ for any $x, y \in S$, $x \neq y$.

Theorem

There is a k-perfect family of functions $[n] \to [k]$ having size $2^{O(k)} \log n$ (and can be constructed in time polynomial in the size of the family).

Instead of trying $O(e^k)$ random colorings, we go through a k-perfect family $\mathcal{H}$ of functions $V(G) \to [k]$.

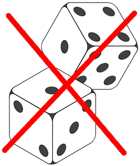
If there is a solution S
$\Rightarrow$ The vertices of S are colorful for at least one $h \in \mathcal{H}$
$\Rightarrow$ Algorithm outputs "YES".
$\Rightarrow$ k-Path can be solved in **deterministic** time $2^{O(k)} \cdot n^{O(1)}$.

Derandomized Color Coding

k-PATH

k-perfect family

$2^{O(k)} \log n$ functions



Finding a colorful path

Solvable in time

$2^k \cdot n^{O(1)}$

Bounded-degree graphs

Meta theorems exist for bounded-degree graphs, but randomization is usually simpler.

Dense k-vertex Subgraph

Input: A graph G, integers k, m.
Find: A set of k vertices inducing $\geq m$ edges.

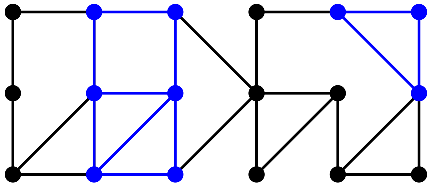
Note: on general graphs, the problem is W[1]-hard parameterized by k, as it contains k-Clique.

Theorem [Cai, Chan, Chan 2006]

Dense k-vertex Subgraph can be solved in randomized time $2^{k(d+1)} \cdot n^{O(1)}$ on graphs with maximum degree d.

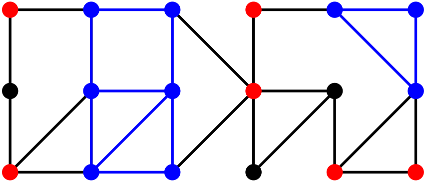
Dense k-vertex Subgraph

- Remove each vertex with probability $1/2$ independently.



Dense k-vertex Subgraph

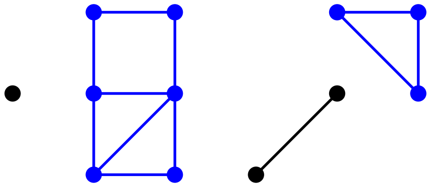
- Remove each vertex with probability $1/2$ independently.



- With probability 2^{-k} no vertex of the solution is removed.
- With probability 2^{-kd} every neighbor of the solution is removed.
- $\Rightarrow$ We have to find a solution that is the union of connected components!

Dense k-vertex Subgraph

- Remove each vertex with probability $1/2$ independently.



- With probability 2^{-k} no vertex of the solution is removed.
- With probability 2^{-kd} every neighbor of the solution is removed.
- $\Rightarrow$ We have to find a solution that is the union of connected components!

Dense k-vertex Subgraph

- Remove each vertex with probability $1/2$ independently.

k_1 vertices	k_2 vertices	k_3 vertices		k_i vertices
m_1 edges	m_2 edges	m_3 edges	$\ldots$	m_i edges

Select connected components with

- at most k vertices and
- at least m edges.

What problem is this?

Dense k-vertex Subgraph

Select connected components with

- at most k vertices and
- at least m edges.

This is exactly KNAPSACK!
(I.e., pick objects of total weight at most S and value at least V.)

We can interpret

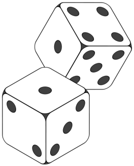
- number of vertices = weight of the items
- number of edges = value of the items

If the weights are integers, then DP solves the problem in time polynomial in the number of objects and the maximum weight.

Dense k-vertex Subgraph

DENSE k-VERTEX SUBGRAPH

Random deletions
success probability:
$2^{-k(d+1)}$



KNAPSACK

Polynomial time

Balanced Separation

Useful problem for recursion:

Balanced Separation

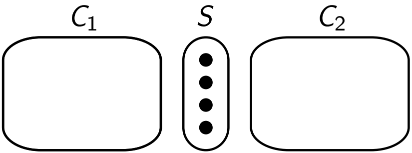
Input: A graph G, integers k, q.

Find: A set S of at most k vertices such that $G \setminus S$ has **two** components of size at least q.

Theorem [Chitnis et al. 2012]

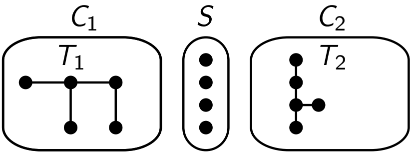
Balanced Separation can be solved in randomized time $2^{O(q+k)} \cdot n^{O(1)}$.

Balanced Separation



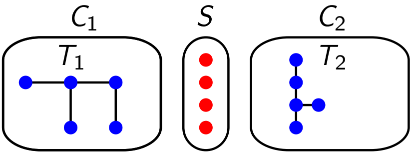
- Remove each vertex with probability $1/2$ independently.

Balanced Separation



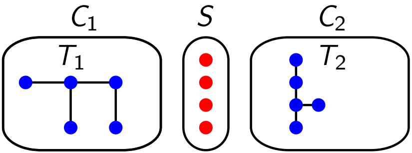
- Remove each vertex with probability $1/2$ independently.

Balanced Separation



- Remove each vertex with probability $1/2$ independently.
- With probability 2^{-k} every vertex of the solution is removed.
- With probability 2^{-q} no vertex of T_1 is removed.
- With probability 2^{-q} no vertex of T_2 is removed.

Balanced Separation

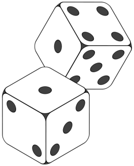


- Remove each vertex with probability $1/2$ independently.
- With probability 2^{-k} every vertex of the solution is removed.
- With probability 2^{-q} no vertex of T_1 is removed.
- With probability 2^{-q} no vertex of T_2 is removed.
- $\Rightarrow$ The reduced graph G' has two components of size $\geq q$ that can be separated in the original graph G by k vertices.
- For any pair of large components of G', we find a minimum $s-t$ cut in G.

Balanced Separation

BALANCED SEPARATION

Random deletions
success probability:
2^{-k+2q}



MINIMUM $s-t$ CUT

Polynomial time

Randomized sampling of important separators

A new technique used by several results:

- MULTICUT [M. and Razgon STOC 2011]
- Clustering problems [Lokshtanov and M. ICALP 2011]
- DIRECTED MULTIWAY CUT [Chitnis, Hajiaghayi, M. SODA 2012]
- DIRECTED MULTICUT in DAGs [Kratsch, Pilipczuk, Pilipczuk, Wahlström ICALP 2012]
- DIRECTED SUBSET FEEDBACK VERTEX SET [Chitnis, Cygan, Hajiaghayi, M. ICALP 2012]
- PARITY MULTIWAY CUT [Lokshtanov, Ramanujan ICALP 2012]
- ... more work in progress.

Transversal problems

Let G be a graph and let $\mathcal{F}$ be a set of subgraphs in G.

Definition

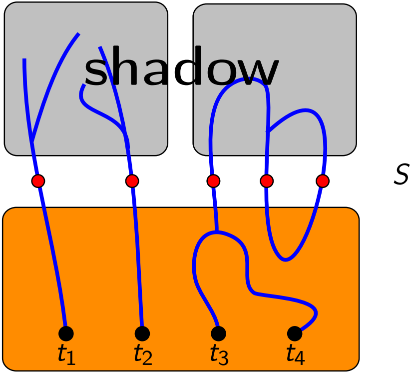
$\mathcal{F}$-transversal: a set of edges of vertices intersecting each subgraph in $\mathcal{F}$ (i.e., "hitting" or "killing" every object in $\mathcal{F}$).

Classical problems formulated as finding a minimum transversal:

- $s - t$ Cut:
  $\mathcal{F}$ is the set of $s - t$ paths.
- Multiway Cut:
  $\mathcal{F}$ is the set of paths between terminals.
- (Directed) Feedback Vertex Set:
  $\mathcal{F}$ is the set of (directed) cycles.
- Delete edges/vertices to make the graph bipartite:
  $\mathcal{F}$ is the set of odd cycles.

The setting

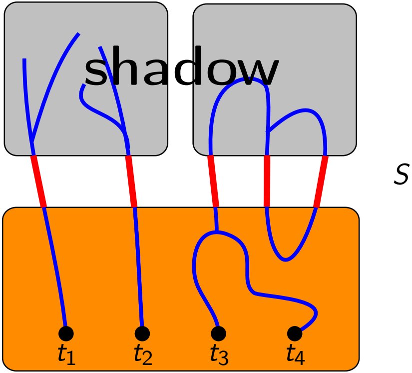
Let $\mathcal{F}$ be a set of connected (not necessarily disjoint!) subgraphs, each intersecting a set T of vertices.



The **shadow** of an $\mathcal{F}$-transversal S is the set of vertices not reachable from T in $G \setminus S$.

The setting

Let $\mathcal{F}$ be a set of connected (not necessarily disjoint!) subgraphs, each intersecting a set T of vertices.



The **shadow** of an $\mathcal{F}$-transversal S is the set of vertices not reachable from T in $G \setminus S$.

The random sampling (undirected edge version)

Shadow: Set of vertices not reachable in $G \setminus S$.

Condition: every $F \in \mathcal{F}$ is connected and intersects T.

Theorem

In $2^{O(k)} \cdot n^{O(1)}$ time, we can compute a set Z with the following property. If there exists an $\mathcal{F}$-transversal of at most k edges, then with probability $2^{-O(k)}$ there is a minimum $\mathcal{F}$-transversal S with

- the shadow of S is covered by Z and
- no edge of S is contained in Z.

Note: The algorithm does not have to know $\mathcal{F}$!

What is this good for?

Clustering

We want to partition objects into clusters subject to certain requirements (typically: related objects are clustered together, bounds on the number or size of the clusters etc.)

(p, q)-CLUSTERING

Input: A graph G, integers p, q.

Find: A partition $(V_1, \ldots, V_m)$ of $V(G)$ such that for every i

- $|V_i| \leq p$ and
- $d(V_i) \leq q$.

$d(V_i)$: number of edges leaving V_i.

Theorem [Lokshtanov and M. 2011]

(p, q)-CLUSTERING can be solved in time $2^{O(q)} \cdot n^{O(1)}$.

A sufficient and necessary condition

Good cluster: size at most p and at most q edges leaving it.

Necessary condition:

Every vertex is contained in a good cluster.

A sufficient and necessary condition

Good cluster: size at most p and at most q edges leaving it.

Necessary condition:

Every vertex is contained in a good cluster.

But surprisingly, this is also a **sufficient condition!**

Lemma

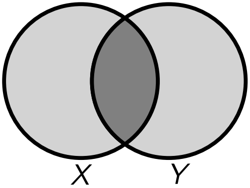
Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

A sufficient and necessary condition

Lemma

Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

Proof: Find a collection of good clusters covering every vertex and having minimum total size. Suppose two clusters intersect.

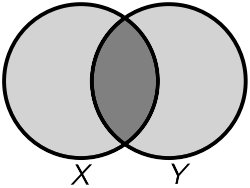


A sufficient and necessary condition

Lemma

Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

Proof: Find a collection of good clusters covering every vertex and having minimum total size. Suppose two clusters intersect.



$$d(X) + d(Y) \geq d(X \setminus Y) + d(Y \setminus X)$$

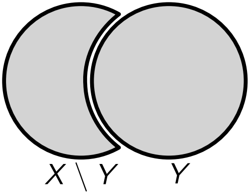
$\Rightarrow$ either $d(X) \geq d(X \setminus Y)$ or $d(Y) \geq d(Y \setminus X)$ holds.

A sufficient and necessary condition

Lemma

Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

Proof: Find a collection of good clusters covering every vertex and having minimum total size. Suppose two clusters intersect.



$$d(X) + d(Y) \geq d(X \setminus Y) + d(Y \setminus X)$$

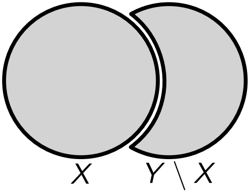
If $d(X) \geq d(X \setminus Y)$, replace X with $X \setminus Y$,
strictly decreasing the total size of the clusters.

A sufficient and necessary condition

Lemma

Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

Proof: Find a collection of good clusters covering every vertex and having minimum total size. Suppose two clusters intersect.



$$d(X) + d(Y) \geq d(X \setminus Y) + d(Y \setminus X)$$

If $d(Y) \geq d(Y \setminus X)$, replace Y with $Y \setminus X$, strictly decreasing the total size of the clusters. QED ■

Finding a good cluster

We have seen:

Lemma

Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

All we have to do is to check if a given vertex v is in a good cluster. Trivial to do in time $n^{O(q)}$.

Finding a good cluster

We have seen:

Lemma

Graph G has a (p, q)-clustering if and only if every vertex is in a good cluster.

All we have to do is to check if a given vertex v is in a good cluster. Trivial to do in time $n^{O(q)}$.

We prove next:

Lemma

We can check in time $2^{O(q)} \cdot n^{O(1)}$ if v is in a good cluster.

This is a transversal problem: we want to hit with q edges every tree going through v and having more than p vertices.

Random sampling (repeated)

Shadow: Set of vertices not reachable in $G \setminus S$.

Condition: every $F \in \mathcal{F}$ is connected and intersects T.

Theorem

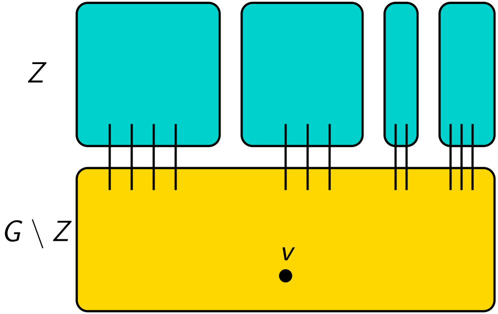
In $2^{O(k)} \cdot n^{O(1)}$ time, we can compute a set Z with the following property. If there exists an $\mathcal{F}$-transversal of at most k edges, then with probability at least $2^{-O(k)}$ there is a minimum $\mathcal{F}$-transversal S with

- the shadow of S is covered by Z and
- no edge of S is contained in Z.

Now:

- $T = \{v\}$
- $\mathcal{F}$ contains every tree going through v having p vertices

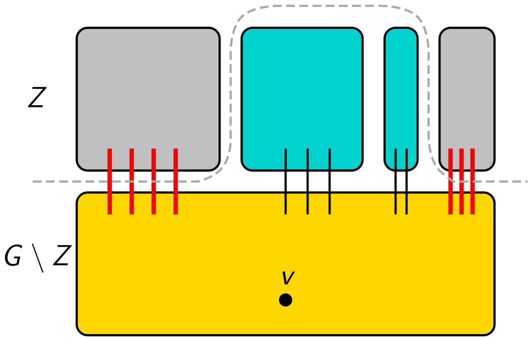
Finding good clusters



- the shadow of S is covered by Z and
- no edge of S is contained in Z.

Where are the edges of S? Where is the good cluster?

Finding good clusters

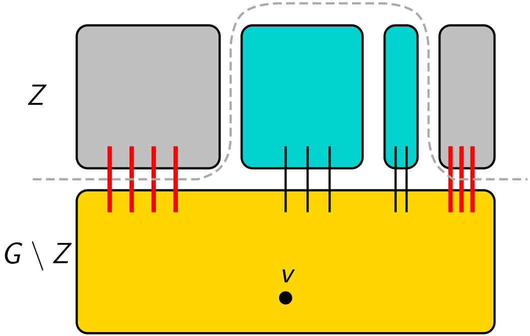


- the shadow of S is covered by Z and
- no edge of S is contained in Z.

Where are the edges of S? Where is the good cluster?

Observe: Components of Z are either fully in the cluster or fully outside the cluster. What is this problem?

Finding good clusters



- the shadow of S is covered by Z and
- no edge of S is contained in Z.

Where are the edges of S? Where is the good cluster?

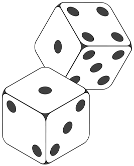
Observe: Components of Z are either fully in the cluster or fully outside the cluster. What is this problem?

KNAPSACK!

(p, q)-CLUSTERING

(p, q)-
CLUSTERING

Random set Z
success probability:
$2^{-O(k)}$



KNAPSACK

Polynomial time

Multiway cut

(Directed) Multiway Cut

Input: Graph G, set of vertices T, integer k

Find: A set S of at most k vertices such that $G \setminus S$ has no (directed) $t_1 - t_2$ path for any $t_1, t_2 \in T$

The undirected version is fairly well understood: best known algorithm solves it in time $2^k \cdot n^{O(1)}$ [Cygan et al. IPEC 2011]

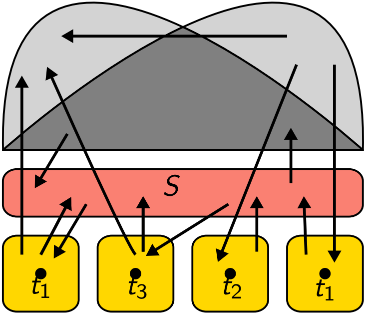
Theorem [Chitnis, Hajiaghayi, Marx 2012]

Directed Multiway Cut is FPT.

Can be formulated as minimum $\mathcal{F}$-transversal, where $\mathcal{F}$ is the set of directed paths between vertices of T.

Directed Multiway Cut

Shadow: those vertices of $G \setminus S$ that cannot be reached from T AND those vertices of $G \setminus S$ from which T cannot be reached.



The random sampling (directed vertex version)

Shadow: those vertices of $G \setminus S$ that cannot be reached from T AND those vertices of $G \setminus S$ from which T cannot be reached.

Condition: for every $F \in \mathcal{F}$ and every vertex $v \in F$, there is a $T \to v$ and a $v \to T$ path in F.

Theorem

In $f(k) \cdot n^{O(1)}$ time, we can compute a set Z with the following property. If there exists an $\mathcal{F}$-transversal of at most k vertices, then with probability $2^{-O(k^2)}$ there is a minimum $\mathcal{F}$-transversal S with

- the shadow of S is covered by Z and
- $S \cap Z = \emptyset$.

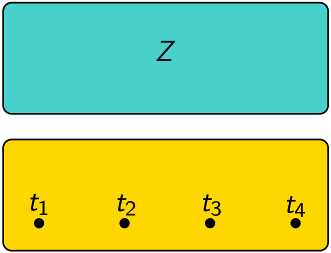
Now:

- T: terminals
- $\mathcal{F}$ contains every directed path between two distinct terminals

Shadow removal

We can assume that Z is disjoint from the solution, so we want to get rid of Z.

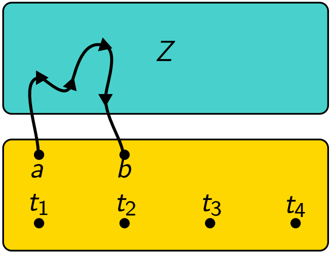
- Deleting Z is not a good idea: can make the problem easier.
- To compensate deleting Z, if there is an $a \to b$ path with internal vertices in Z, add a direct $a \to b$ edge.



Shadow removal

We can assume that Z is disjoint from the solution, so we want to get rid of Z.

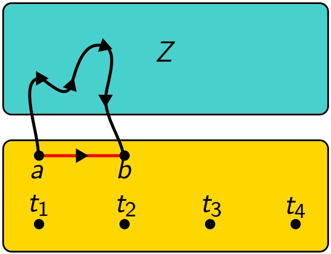
- Deleting Z is not a good idea: can make the problem easier.
- To compensate deleting Z, if there is an $a \to b$ path with internal vertices in Z, add a direct $a \to b$ edge.



Shadow removal

We can assume that Z is disjoint from the solution, so we want to get rid of Z.

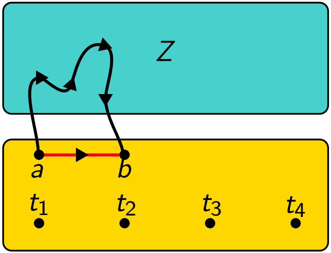
- Deleting Z is not a good idea: can make the problem easier.
- To compensate deleting Z, if there is an $a \to b$ path with internal vertices in Z, add a direct $a \to b$ edge.



Shadow removal

We can assume that Z is disjoint from the solution, so we want to get rid of Z.

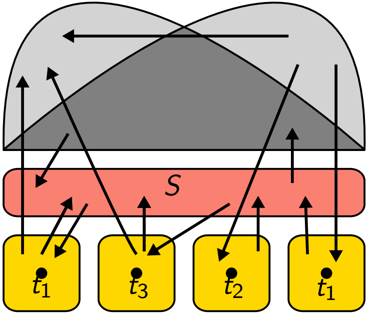
- Deleting Z is not a good idea: can make the problem easier.
- To compensate deleting Z, if there is an $a \to b$ path with internal vertices in Z, add a direct $a \to b$ edge.



Crucial observation:
S remains a solution (since Z is disjoint from S) and
S is a **shadowless solution** (since Z covers the shadow of S).

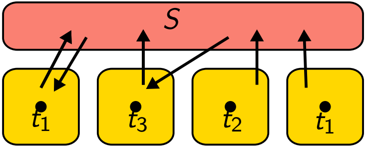
Shadowless solutions

How does a shadowless solution look like?



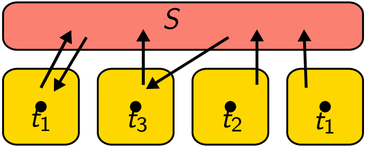
Shadowless solutions

How does a shadowless solution look like?



Shadowless solutions

How does a shadowless solution look like?

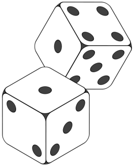


It is an undirected multiway cut in the underlying undirected graph!
$\Rightarrow$ **Problem can be reduced to undirected multiway cut.**

Directed Multiway Cut

DIRECTED MULTIWAY CUT

Random set Z
success probability:
$2^{-O(k^2)}$



UNDIRECTED MULTIWAY CUT

$2^k \cdot n^{O(1)}$ time

Cut and count

A very powerful technique for many problems on graphs of bounded-treewidth.

Classical result:

Theorem

Given a tree decomposition of width k, Hamiltonian Cycle can be solved in time $k^{O(k)} \cdot n^{O(1)} = 2^{O(k \log k)} \cdot n^{O(1)}$.

Very recently:

Theorem [Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, Wojtaszczyk 2011]

Given a tree decomposition of width k, Hamiltonian Cycle can be solved in time $4^k \cdot n^{O(1)}$.

Isolation Lemma

Isolation Lemma [Mulmuley, Vazirani, Vazirani 1987]

Let $\mathcal{F}$ be a nonempty family of subsets of U and assign a weight $w(u) \in [N]$ to each $u \in U$ uniformly and independently at random. The probability that there is a **unique** $S \in \mathcal{F}$ having minimum weight is at least

$$1 - \frac{|U|}{N}.$$

Isolation Lemma

Isolation Lemma [Mulmuley, Vazirani, Vazirani 1987]

Let $\mathcal{F}$ be a nonempty family of subsets of U and assign a weight $w(u) \in [N]$ to each $u \in U$ uniformly and independently at random. The probability that there is a **unique** $S \in \mathcal{F}$ having minimum weight is at least

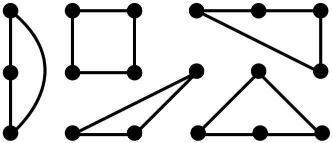
$$1 - \frac{|U|}{N}.$$

Let $U = E(G)$ and $\mathcal{F}$ be the set of all Hamiltonian cycles.

- By setting $N := |V(G)|^{O(1)}$, we can assume that there is a unique minimum weight Hamiltonian cycle.
- If N is polynomial in the input size, we can guess this minimum weight.
- So we are looking for a Hamiltonian cycle of weight **exactly** C, under the assumption that there is a **unique** such cycle.

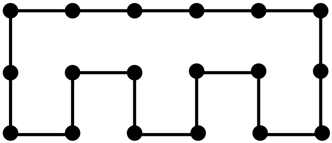
Cycle covers

- **Cycle cover:** A subgraph having degree exactly two at each vertex.



Cycle covers

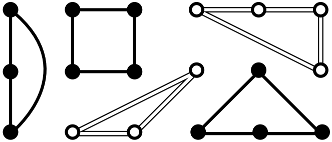
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- A Hamiltonian cycle is a cycle cover, but a cycle cover can have more than one component.

Cycle covers

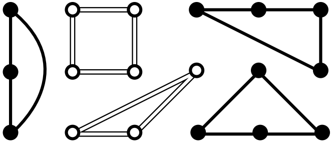
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- A Hamiltonian cycle is a cycle cover, but a cycle cover can have more than one component.
- **Colored cycle cover:** each component is colored black or white.

Cycle covers

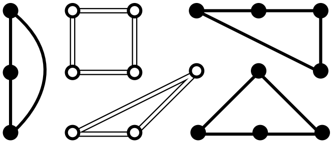
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- A Hamiltonian cycle is a cycle cover, but a cycle cover can have more than one component.
- **Colored cycle cover:** each component is colored black or white.

Cycle covers

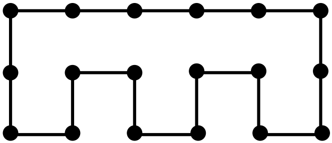
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- A Hamiltonian cycle is a cycle cover, but a cycle cover can have more than one component.
- **Colored cycle cover:** each component is colored black or white.
- A cycle cover with k components gives rise to 2^k colored cycle covers.
  - If there is no weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is 0 mod 4.
  - If there is a unique weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is 2 mod 4.

Cycle covers

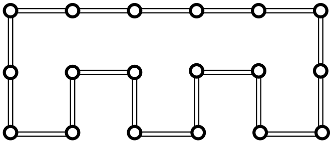
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- A Hamiltonian cycle is a cycle cover, but a cycle cover can have more than one component.
- **Colored cycle cover:** each component is colored black or white.
- A cycle cover with k components gives rise to 2^k colored cycle covers.
  - If there is no weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is 0 mod 4.
  - If there is a unique weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is 2 mod 4.

Cycle covers

- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- A Hamiltonian cycle is a cycle cover, but a cycle cover can have more than one component.
- **Colored cycle cover:** each component is colored black or white.
- A cycle cover with k components gives rise to 2^k colored cycle covers.
    - If there is no weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is 0 mod 4.
    - If there is a unique weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is 2 mod 4.

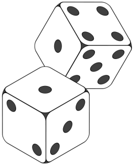
Cut and Count

- Assign random weights $\leq 2|E(G)|$ to the edges.
- If there is a Hamiltonian cycle, then with probability $1/2$, there is a C such that there is a **unique** weight-C Hamiltionian cycle.
- Try all possible C.
- Count the number of weight-C colored cycle covers: can be done in time $4^k \cdot n^{O(1)}$ if a tree decomposition of width k is given.
- Answer YES if this number is 2 mod 4.

Cut and Count

HAMILTONIAN CYCLE

Random weights success probability: $1/2$



Counting weighted colored cycle covers

$4^k \cdot n^{O(1)}$ time

Conclusions

- Randomization gives elegant solution to many problems.
- Derandomization is sometimes possible (but less elegant).
- Small (but $f(k)$) success probability is good for us.
- Reducing the problem we want to solve to a problem that is easier to solve.