Parameterized Algorithms
Introduction II

Lecture #2
October 26, 2021

Recap: fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Recap: fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Examples of NP-hard problems that are FPT:

- Finding a vertex cover of size k.
- Finding a path of length k.
- Finding k disjoint triangles.
- Drawing the graph in the plane with k edge crossings.
- Finding disjoint paths that connect k pairs of points.
- ...

Recap: fixed-parameter tractability

Main definition

A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Main questions:

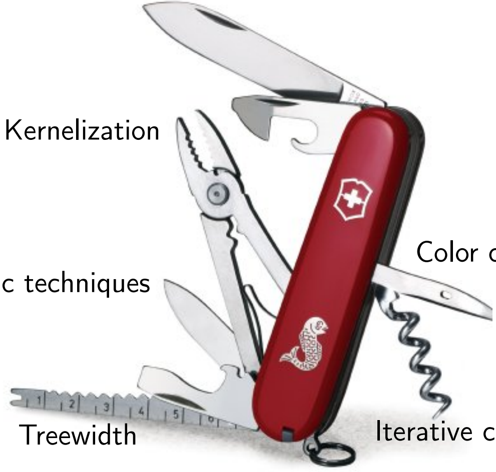
- Is the problem fixed-parameter tractable (FPT) with a given parameter?
- What is the best possible $f(k)$ in the running time?

Recap: FPT techniques

Bounded-depth search trees

Kernelization

Algebraic techniques



Color coding

Treewidth

Iterative compression

Recap: branching

Idea: reduce the problem into a bounder number of instances with strictly smaller parameter.

- Branching into c directions: $O^*(c^k)$ algorithms.
- Branching into k directions: $O^*(k^k)$ algorithms.
- Branching vectors and analysis of recurrences of the form

$$T(k) \leq T(k-1) + 2T(k-2) + T(k-3)$$

- Graph modification problems where the graph property can be characterized by a finite set of forbidden induced subgraphs is FPT.

Closest String

> **Closest String**
> Given strings $s_1, \dots, s_k$ of length L over alphabet Σ, and an integer d, find a string s (of length L) such that Hamming distance $d(s, s_i) \le d$ for every $1 \le i \le k$.

(Hamming distance: number of differing positions)

s_1	C	B	D	C	C	A	C	B	B
s_2	A	B	D	B	C	A	B	D	B
s_3	C	D	D	B	A	C	C	B	D
s_4	D	D	A	B	A	C	C	B	D
s_5	A	C	D	B	D	D	C	B	C

Closest String

Closest String

Given strings $s_1, \ldots, s_k$ of length L over alphabet Σ, and an integer d, find a string s (of length L) such that Hamming distance $d(s, s_i) \le d$ for every $1 \le i \le k$.

(Hamming distance: number of differing positions)

s_1	C	B	D	C	C	A	C	B	B
s_2	A	B	D	B	C	A	B	D	B
s_3	C	D	D	B	A	C	C	B	D
s_4	D	D	A	B	A	C	C	B	D
s_5	A	C	D	B	D	D	C	B	C
	A	D	D	B	C	A	C	B	D

Closest String

Closest String

Given strings $s_1, \ldots, s_k$ of length L over alphabet Σ, and an integer d, find a string s (of length L) such that Hamming distance $d(s, s_i) \le d$ for every $1 \le i \le k$.

(Hamming distance: number of differing positions)

s_1	C	B	D	C	C	A	C	B	B
s_2	A	B	D	B	C	A	B	D	B
s_3	C	D	D	B	A	C	C	B	D
s_4	D	D	A	B	A	C	C	B	D
s_5	A	C	D	B	D	D	C	B	C
	A	D	D	B	C	A	C	B	D

Different parameters:

- Number k of strings.
- Length L of strings.
- Maximum distance d.
- Alphabet size $|\Sigma|$.

Closest String

> **Closest String**
> Given strings $s_1, \ldots, s_k$ of length L over alphabet Σ, and an integer d, find a string s (of length L) such that Hamming distance $d(s, s_i) \leq d$ for every $1 \leq i \leq k$.

(Hamming distance: number of differing positions)

s_1	C	B	D	C	C	A	C	B	B
s_2	A	B	D	B	C	A	B	D	B
s_3	C	D	D	B	A	C	C	B	D
s_4	D	D	A	B	A	C	C	B	D
s_5	A	C	D	B	D	D	C	B	C
	A	D	D	B	C	A	C	B	D

Different parameters:

- Number k of strings.
- Length L of strings.
- Maximum distance d.
- Alphabet size $|\Sigma|$.

We can ask for running time for example

- $f(d)n^{O(1)}$: FPT parameterized by d
- $f(k, |\Sigma|)n^{O(1)}$: FPT with combined parameters k and $|\Sigma|$

Closest String

Note: Taking the majority at each position is in general *not* the best solution.

s_1	A	A	A	A	A	A	
s_2	B	B	B	B	B	B	
s_3	B	B	B	B	B	B	
s_4	B	B	B	B	B	B	
s_5	B	B	B	B	B	B	
majority	B	B	B	B	B	B	– distance 6 from s_1
opt	A	A	A	B	B	B	– distance 3 from every s_i

The positions are not independent!

Closest String

Theorem

Closest String can be solved in time $2^{O(d \log d)} n^{O(1)}$.

- **Main idea:** Given a string y at Hamming distance ℓ from some solution, we use branching to find a string at distance at most $\ell - 1$ from some solution.
- Initially, $y = x_1$ is at distance at most d from some solution.

Closest String

Theorem

Closest String can be solved in time $2^{O(d \log d)} n^{O(1)}$.

- **Main idea:** Given a string y at Hamming distance ℓ from some solution, we use branching to find a string at distance at most $\ell - 1$ from some solution.
- Initially, $y = x_1$ is at distance at most d from some solution.
- If y is not a solution, then there is an x_i with $d(y, x_i) \geq d + 1$.
  - Look at the first $d + 1$ positions p where $x_i[p] \neq y[p]$. For every solution z, it is true for one such p that $x_i[p] = z[p]$.
  - Branch on choosing one of these $d + 1$ positions and replace $y[p]$ with $x_i[p]$: distance of y from solution z decreases to $\ell - 1$.
- Running time $(d + 1)^d \cdot n^{O(1)} = 2^{O(d \log d)} n^{O(1)}$.

Branching: wrap up

- Branching into c directions: $O^*(c^k)$ algorithms.
- Branching into k directions: $O^*(k^k)$ algorithms.
- Branching vectors and analysis of recurrences of the form

$$T(k) \leq T(k-1) + 2T(k-2) + T(k-3)$$

- Graph modification problems where the graph property can be characterized by a finite set of forbidden induced subgraphs is FPT.

Kernelization

Data reductions

We would like to efficiently reduce the input size of a hard problem to make it more tractable.

Data reductions

We would like to efficiently reduce the input size of a hard problem to make it more tractable.

> Is there a polynomial-time algorithm that *always* reduces the size of the input by 1?

Data reductions

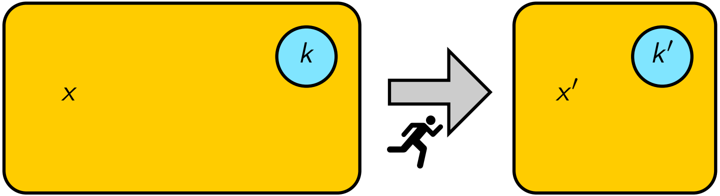
We would like to efficiently reduce the input size of a hard problem to make it more tractable.

Is there a polynomial-time algorithm that *always* reduces the size of the input by 1?

Obviously, only if the problem is polynomial-time solvable.

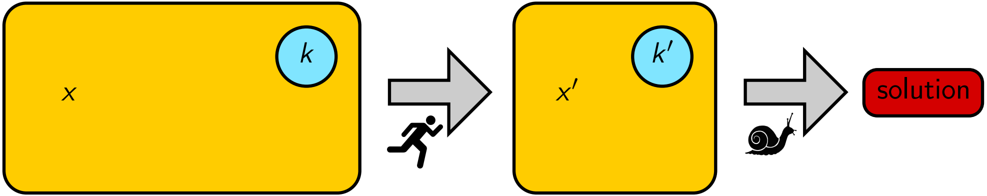
Data reductions—with a guarantee

- **Kernelization** is a method for parameterized preprocessing:
  - We want to efficiently reduce the size of the instance (x, k) to an equivalent instance with size bounded by $f(k)$.
- A basic way of obtaining FPT algorithms:
  - Reduce the size of the instance to $f(k)$ in polynomial time and then apply any brute force algorithm to the shrunk instance.
- Kernelization is also a rigorous mathematical analysis of efficient preprocessing.



Data reductions—with a guarantee

- **Kernelization** is a method for parameterized preprocessing:
  - We want to efficiently reduce the size of the instance (x, k) to an equivalent instance with size bounded by $f(k)$.
- A basic way of obtaining FPT algorithms:
  - Reduce the size of the instance to $f(k)$ in polynomial time and then apply any brute force algorithm to the shrunk instance.
- Kernelization is also a rigorous mathematical analysis of efficient preprocessing.



Kernel for Vertex Cover

Reduction rules for instance (G, k):

(R1) If v is an isolated vertex, then reduce to $(G - v, k)$.

(R2) If v has degree more than k, then reduce to $(G - v, k - 1)$.

Kernel for VERTEX COVER

Reduction rules for instance (G, k):

(R1) If v is an isolated vertex, then reduce to $(G - v, k)$.

(R2) If v has degree more than k, then reduce to $(G - v, k - 1)$.

Lemma

If (G, k) is a yes-instance of VERTEX COVER such that (R1) and (R2) cannot be applied, then $|E(G)| \leq k^2$ and $|V(G)| \leq k^2 + k$.

Kernel for VERTEX COVER

Reduction rules for instance (G, k):

(R1) If v is an isolated vertex, then reduce to $(G - v, k)$.

(R2) If v has degree more than k, then reduce to $(G - v, k - 1)$.

Lemma

If (G, k) is a yes-instance of VERTEX COVER such that (R1) and (R2) cannot be applied, then $|E(G)| \leq k^2$ and $|V(G)| \leq k^2 + k$.

Proof:

- Each of the k vertices of the solution can cover at most k edges (by (R2)).
- Every vertex of G is either in the solution, or one of the $\leq k$ neighbors of a vertex in a solution (by (R1)+(R2)).

Kernel for Vertex Cover

Reduction rules for instance (G, k):

(R1) If v is an isolated vertex, then reduce to $(G - v, k)$.

(R2) If v has degree more than k, then reduce to $(G - v, k - 1)$.

Lemma

If (G, k) is a yes-instance of Vertex Cover such that (R1) and (R2) cannot be applied, then $|E(G)| \leq k^2$ and $|V(G)| \leq k^2 + k$.

Kernelization for Vertex Cover:

- Apply rules (R1) and (R2) exhaustively.
- If $|E(G)| > k^2$ or $|V(G)| > k^2 + k$, then we have a no-instance.
- Otherwise, we have a kernel of size $O(k^2)$.

Kernelization: formal definition

- Let $P \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem and $f : \mathbb{N} \to \mathbb{N}$ a computable function.
- A **kernel** for P of size f is an algorithm that, given (x, k), takes time polynomial in $|x| + k$ and outputs an instance (x', k') such that
  - $(x, k) \in P \iff (x', k') \in P$
  - $|x'| \leq f(k)$, $k' \leq f(k)$.
- A **polynomial kernel** is a kernel whose function f is polynomial.

Kernelization: formal definition

- Let $P \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem and $f : \mathbb{N} \to \mathbb{N}$ a computable function.
- A **kernel** for P of size f is an algorithm that, given (x, k), takes time polynomial in $|x| + k$ and outputs an instance (x', k') such that
  - $(x, k) \in P \iff (x', k') \in P$
  - $|x'| \leq f(k)$, $k' \leq f(k)$.
- A **polynomial kernel** is a kernel whose function f is polynomial.

Which parameterized problems have kernels?

A surprising equivalence

Theorem

A parameterized problem is FPT if and only if it is decidable and has a kernel (of arbitrary size).

A surprising equivalence

Theorem

A parameterized problem is FPT if and only if it is decidable and has a kernel (of arbitrary size).

Proof:

- If the problem has a kernel:
  Reducing the size of the instance to $f(k)$ in poly time + brute force
  $\Rightarrow$ problem is FPT.

A surprising equivalence

Theorem

A parameterized problem is FPT if and only if it is decidable and has a kernel (of arbitrary size).

Proof:

- If the problem has a kernel:
  Reducing the size of the instance to $f(k)$ in poly time + brute force
  $\Rightarrow$ problem is FPT.
- If the problem can be solved in time $f(k)|x|^{O(1)}$:
  - If $|x| \leq f(k)$, then we already have a kernel of size $f(k)$.
  - If $|x| \geq f(k)$, then we can solve the problem in time $f(k)|x|^{O(1)} \leq |x| \cdot |x|^{O(1)}$ (polynomial in $|x|$) and then output a trivial yes- or no-instance.

A surprising equivalence

Theorem

A parameterized problem is FPT if and only if it is decidable and has a kernel (of arbitrary size).

Proof:

- If the problem has a kernel:
  Reducing the size of the instance to $f(k)$ in poly time + brute force
  $\Rightarrow$ problem is FPT.
- If the problem can be solved in time $f(k)|x|^{O(1)}$:
  - If $|x| \leq f(k)$, then we already have a kernel of size $f(k)$.
  - If $|x| \geq f(k)$, then we can solve the problem in time $f(k)|x|^{O(1)} \leq |x| \cdot |x|^{O(1)}$ (polynomial in $|x|$) and then output a trivial yes- or no-instance.
- The existence of kernels is not a separate question...
- ...but the existence of **polynomial kernels** is a deep and nontrivial topic!

Kernelization: summary

- Can we efficiently preprocess the input to reduce the size to $f(k)$?
- We have seen: a kernel of size $O(k^2)$ for Vertex Cover.
- Kernelization follows from FPT algorithm, but the existence of a **polynomial kernel** is a separate question.
- There are problems where e.g. branching immediately gives an FPT algorithm, but this does not give a polynomial kernel.
- Later:
  - Sunflower Lemma
  - 2-Expansion Lemma
  - Crown Decomposition
  - Linear Programming
- Lower bounds

Color Coding

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 randomization

Polynomial-time randomized algorithms

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

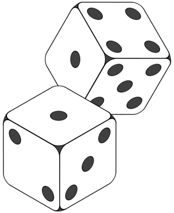
Randomized FPT algorithms

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

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 on k vertices.

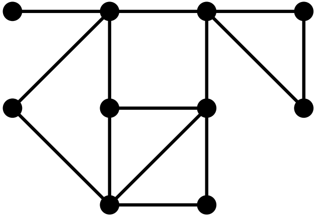
Note: The problem is clearly NP-hard, as it contains the Hamiltonian Path problem. But finding a *walk* is easy.

Theorem

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

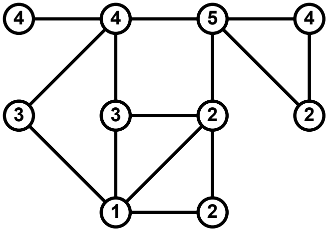
Color Coding

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



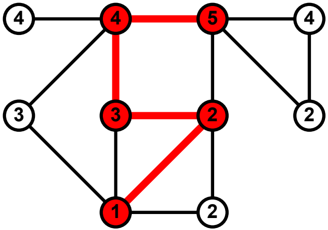
Color Coding

- 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

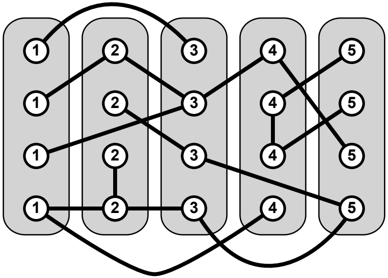
Useful fact

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

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

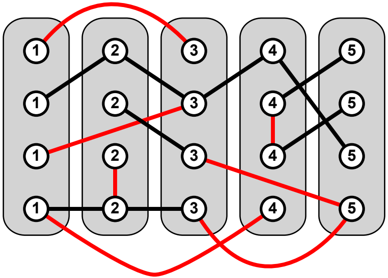
- Thus if $p > k^{-k}$, then error probability is at most $1/e$ after k^k repetitions.
- Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- For example, by trying $100 \cdot k^k$ random colorings, the probability of a wrong answer is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

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



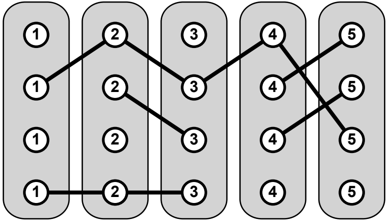
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

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



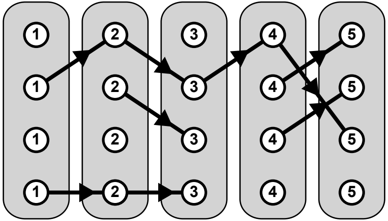
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

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



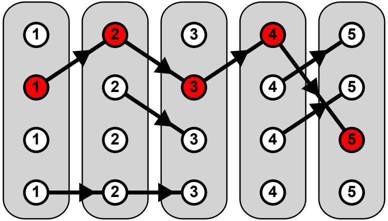
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

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 $1 - 2 - \cdots - k$

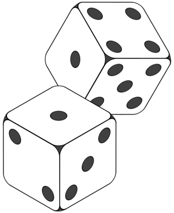


- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Color Coding

k-**PATH**

Color Coding
success probability: k^{-k}

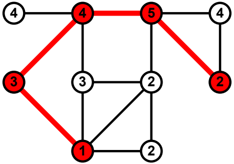


Finding a $1 - 2 - \cdots - k$ **colored path**

polynomial-time solvable

Improved Color Coding

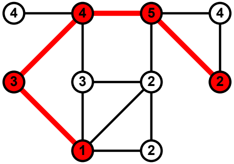
- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- 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.



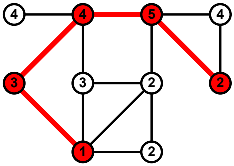
- Check if there is a **colorful** path where each color appears exactly once on the vertices; output "YES" or "NO".
  - If there is no k-path: no **colorful** path exists $\Rightarrow$ "NO".
  - If there is a k-path: the probability that it is **colorful** is

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

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 path P 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.

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

Initialization:

For every v with color r, $x(v, \{r\}) = \text{TRUE}$.

Recurrence:

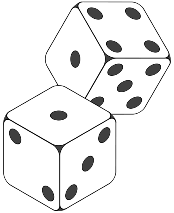
For every v with color r and set $C \subseteq [k]$

$$x(v, C) = \bigvee_{u \in N(v)} x(u, C \setminus \{r\}).$$

Improved Color Coding

k-PATH

Color Coding
success probability: e^{-k}



Finding a colorful path

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

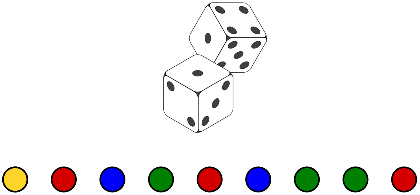
Derandomization

De-randomization: removing the random choices, making the algorithm deterministic.

Derandomization

De-randomization: removing the random choices, making the algorithm deterministic.

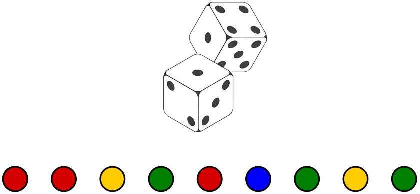
Randomized



Derandomization

De-randomization: removing the random choices, making the algorithm deterministic.

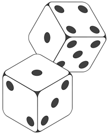
Randomized



Derandomization

De-randomization: removing the random choices, making the algorithm deterministic.

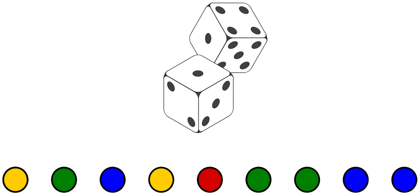
Randomized



Derandomization

De-randomization: removing the random choices, making the algorithm deterministic.

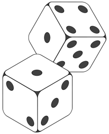
Randomized



Derandomization

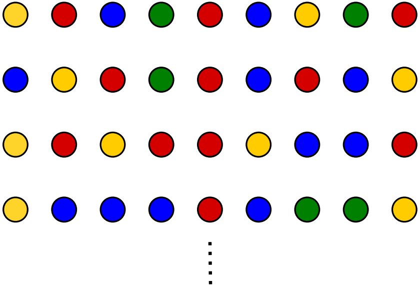
De-randomization: removing the random choices, making the algorithm deterministic.

Randomized



Deterministic

fixed family of colorings



Instead of repeatedly using randomness, we go through a special family of colorings.

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 $V(G) \to [k]$.

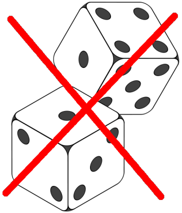
If there is a solution S
$\Rightarrow$ The vertices of S are colorful for at least one $h \in \mathcal{H}$
$\Rightarrow$ Algorithm outputs "YES".
$\Rightarrow$ k-Path can be solved in **deterministic** time $2^{O(k)} \cdot n^{O(1)}$.

Derandomized Color Coding

k-**PATH**

k-perfect family

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



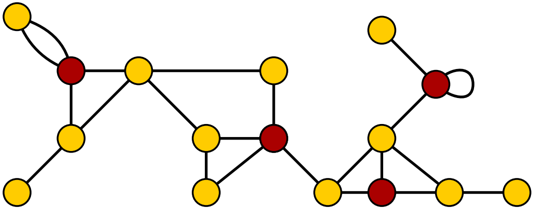
Finding a colorful path

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

Recap: Feedback Vertex Set

Feedback Vertex Set:
Given (G, k), find a set S of at most k vertices such that $G - S$ has no cycles.

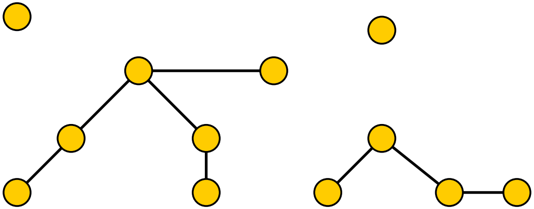
- We allow multiple parallel edges and self loops.
- A **feedback vertex set** is a set that hits every cycle in the graph.



Recap: Feedback Vertex Set

> **Feedback Vertex Set:**
> Given (G, k), find a set S of at most k vertices such that $G - S$ has no cycles.

- We allow multiple parallel edges and self loops.
- A **feedback vertex set** is a set that hits every cycle in the graph.



Recap: Feedback Vertex Set

- If we find a cycle, then we have to include at least one of its vertices into the solution. But the length of the cycle can be arbitrary large!
- **Main idea:** We identify a set of $O(k)$ vertices such that any size-k feedback vertex set has to contain one of these vertices.
- But first: some reductions to simplify the problem.

Reduction rules

(R1) If there is a loop at v, then delete v and decrease k by one.

(R2) If there is an edge of multiplicity larger than 2, then reduce its multiplicity to 2.

(R3) If there is a vertex v of degree at most 1, then delete v.

(R4) If there is a vertex v of degree 2, then delete v and add an edge between the neighbors of v.

If the reduction rules cannot be applied, then every vertex has degree at least 3.

Recap: Branching for Feedback Vertex Set

Let G be a graph whose vertices have degree at least 3.

- Order the vertices as v_1, v_2, ..., v_n by **decreasing** degree (breaking ties arbitrarily).
- Let $V_{3k} = \{v_1, \ldots, v_{3k}\}$ be the $3k$ largest-degree vertices.

Lemma

If G has minimum degree at least 3, then every feedback vertex set S of size at most k contains a vertex from V_{3k}.

Recap: Branching for Feedback Vertex Set

Let G be a graph whose vertices have degree at least 3.

- Order the vertices as v_1, v_2, ..., v_n by **decreasing** degree (breaking ties arbitrarily).
- Let $V_{3k} = \{v_1, \ldots, v_{3k}\}$ be the $3k$ largest-degree vertices.

Lemma

If G has minimum degree at least 3, then every feedback vertex set S of size at most k contains a vertex from V_{3k}.

Algorithm:

- Apply the reduction rules (poly time) $\Rightarrow$ graph has minimum degree 3.
- For each vertex $v \in V_{3k}$, recurse on the instance $(G - v, k - 1)$.
- Running time $(3k)^k \cdot n^{O(1)} = 2^{O(k \log k)} \cdot n^{O(1)}$.

Randomized algorithm for Feedback Vertex Set

Identifying a vertex of the solution randomly:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Randomized algorithm for Feedback Vertex Set

Identifying a vertex of the solution randomly:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Consequence: if we select a random edge uv and select a random endpoint $x \in \{u, v\}$, then x is in some solution S with probability at least $1/4$.

Randomized algorithm for FEEDBACK VERTEX SET

Identifying a vertex of the solution randomly:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Consequence: if we select a random edge uv and select a random endpoint $x \in \{u, v\}$, then x is in some solution S with probability at least $1/4$.

Algorithm for finding a solution of size k with probability $\geq 4^{-k}$:

- Apply reductions.
- Select random edge and random endpoint x.
- Remove x.
- Recurse with parameter $k - 1$.

Randomized algorithm for FEEDBACK VERTEX SET

Identifying a vertex of the solution randomly:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Consequence: if we select a random edge uv and select a random endpoint $x \in \{u, v\}$, then x is in some solution S with probability at least $1/4$.

Algorithm for finding a solution of size k with probability $\geq 4^{-k}$:

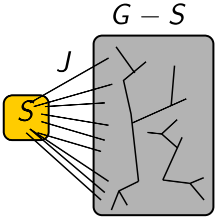
- Apply reductions.
- Select random edge and random endpoint x. $\Rightarrow$ good with prob. $\geq 1/4$
- Remove x.
- Recurse with parameter $k-1$. $\Rightarrow$ good with prob. $\geq 4^{-(k-1)}$

Note: $1/4 \cdot 4^{-(k-1)} = 4^{-k}$.

Proof of lemma:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

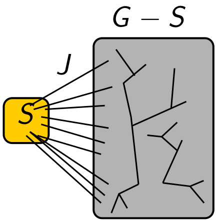


Proof of lemma:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Only the edges in $G - S$ are BAD $\Rightarrow < |V(G-S)|$ BAD edges.



Proof of lemma:

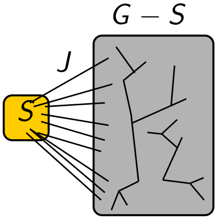
Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Only the edges in $G - S$ are BAD $\Rightarrow < |V(G-S)|$ BAD edges.

Every edge in J is GOOD, lower bound on their number:

- Classify the vertices of $G - S$ into $V_{\leq 1}$, $V_{=2}$, $V_{>2}$ by degree.
- Each vertex in $V_{\leq 1}$ contributes ≥ 2 edges to J.
- Each vertex in $V_{=2}$ contributes ≥ 1 edges to J.

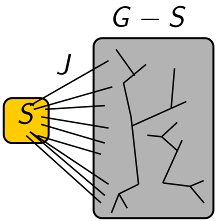


Proof of lemma:

Lemma

Let G be a graph with minimum degree at least 3 and let S be a feedback vertex set of G. Then more than half of the edges have at least on endpoint in S.

Only the edges in $G - S$ are BAD $\Rightarrow < |V(G-S)|$ BAD edges.



Every edge in J is GOOD, lower bound on their number:

- Classify the vertices of $G - S$ into $V_{\leq 1}$, $V_{=2}$, $V_{>2}$ by degree.
- Each vertex in $V_{\leq 1}$ contributes ≥ 2 edges to J.
- Each vertex in $V_{=2}$ contributes ≥ 1 edges to J.
- Number of GOOD edges is more than the number of BAD edges:

$$2|V_{\leq 1}| + |V_{=2}| > |V_{\leq 1}| + |V_{=2}| + |V_{>2}| = |V(G-S)|$$

($|V_{\leq 1}| > |V_{>2}|$ because $G - S$ is a forest)

Summary

- **Questions**
  - Is the problem fixed-parameter tractable (FPT) with a given parameter?
  - What is the best possible $f(k)$ in the running time?
  - Is there a polynomial kernel?
- **Branching**
  - $2^{O(k)} \cdot n^{O(1)}$ time algorithms for Vertex Cover and Triangle Free Deletion.
  - $2^{O(k \log k)} n^{O(1)}$ time algorithms for Feedback Vertex Set and Closest String
- **Kernelization**
  - $O(k^2)$ kernel for Vertex Cover.
- **Randomization**
  - $2^{O(k)} \cdot n^{O(1)}$ (randomized) algorithm for k-Path using Color Coding.
  - $4^k \cdot n^{O(1)}$ (randomized) algorithm for Feedback Vertex Set.

The race for better FPT algorithms