Fine-Grained Complexity and Algorithm Design Boot Camp

Recent Advances in FPT and Exact Algorithms for NP-Complete Problems

Dániel Marx

Institute for Computer Science and Control,
Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

Simons Institute, Berkeley, CA
September 1, 2015

Overview

- Today:
  Introduction to FPT, classical and more recent examples.
  - Definition of FPT.
  - Simple classical examples.
  - Treewidth.
  - Algorithms and applications of treewidth.
- Wednesday 3pm:
  Parameterized reductions — negative evidence for FPT.
- Thursday 3pm:
  (Tight) lower bounds based on ETH.
- Friday 3pm:
  (Even tighter) lower bounds based on SETH.

Parameterized problems

Main idea

Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

Parameterized problems

Main idea

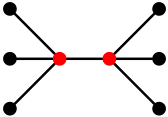
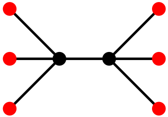
Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

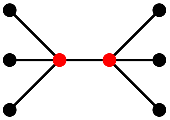
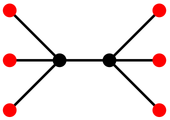
What can be the parameter k?

- The size k of the solution we are looking for.
- The maximum degree of the input graph.
- The dimension of the point set in the input.
- The length of the strings in the input.
- The length of clauses in the input Boolean formula.
- $\dots$

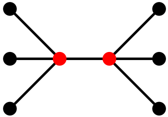
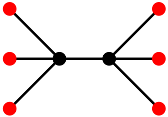
Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete

Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities
	$O(2^k n^2)$ **algorithm exists** **exists**	**No** $n^{o(k)}$ **algorithm known**

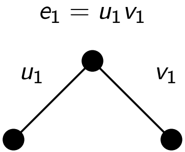
Bounded search tree method

Algorithm for VERTEX COVER:

$e_1 = u_1 v_1$

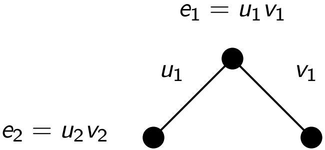
Bounded search tree method

Algorithm for Vertex Cover:



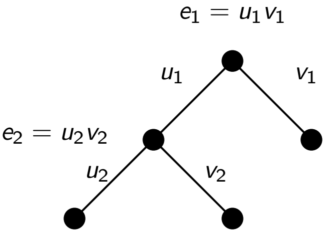
Bounded search tree method

Algorithm for VERTEX COVER:



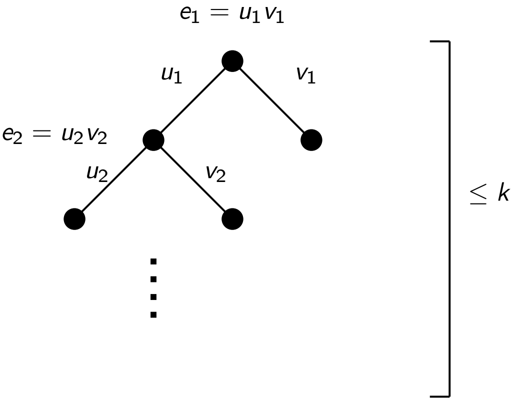
Bounded search tree method

Algorithm for VERTEX COVER:



Bounded search tree method

Algorithm for VERTEX COVER:



Height of the search tree $\leq k$ $\Rightarrow$ at most 2^k leaves $\Rightarrow$ $2^k \cdot n^{O(1)}$ time algorithm.

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.

Fixed-parameter tractability

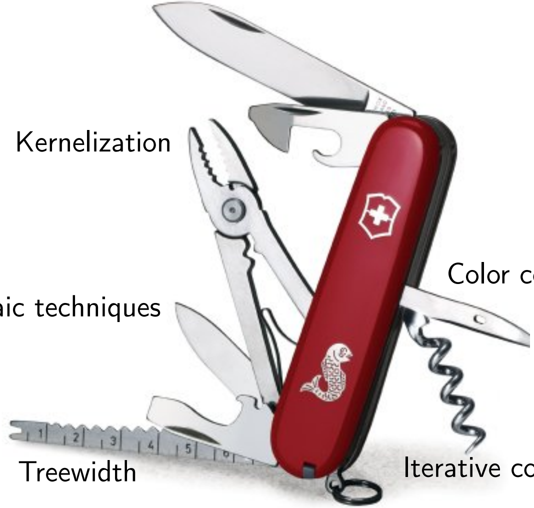
Main definition

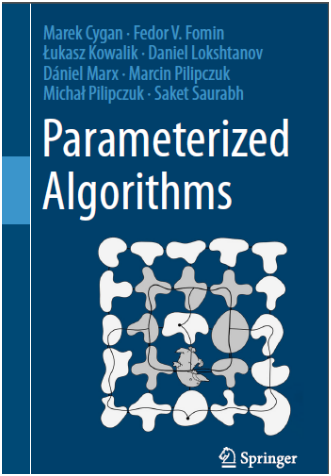
A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

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.
- ...

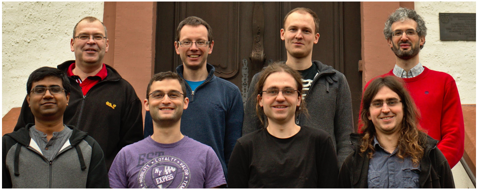
FPT techniques





Parameterized Algorithms

Marek Cygan, Fedor V. Fomin,
Lukasz Kowalik, Daniel Lokshtanov,
Dániel Marx, Marcin Pilipczuk,
Michał Pilipczuk, Saket Saurabh



W[1]-hardness

Negative evidence similar to NP-completeness. If a problem is **W[1]-hard,** then the problem is not FPT unless FPT=W[1].

Some W[1]-hard problems:

- Finding a clique/independent set of size k.
- Finding a dominating set of size k.
- Finding k pairwise disjoint sets.
- ...

More about this on Wednesday at 3pm.

Games to play

- The FPT vs. W[1]-hard game

  Is the problem fixed-parameter tractable?
- The $f(k)$ game for FPT problems

  What is the best $f(k)$ dependence on the parameter?
- The exponent game for W[1]-hard problems

  What is the best possible dependence on k in the exponent?

Significant progress on these questions in recent years, both from the algorithmic and from the complexity side.

Color coding

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.

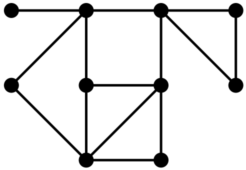
Theorem [Alon, Yuster, Zwick 1994]

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

Previous best algorithms had running time $k^{O(k)} \cdot n^{O(1)}$.

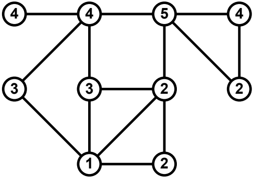
Color Coding

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



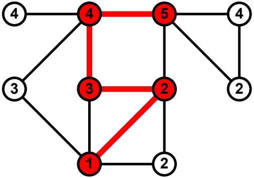
Color Coding

- 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

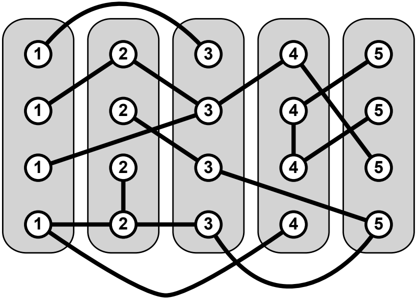
Useful fact

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

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

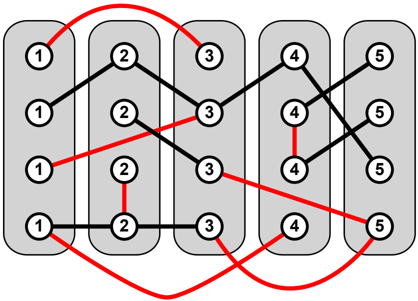
- Thus if $p > k^{-k}$, then error probability is at most $1/e$ after k^k repetitions.
- Repeating the whole algorithm a constant number of times can make the error probability an arbitrary small constant.
- For example, by trying $100 \cdot k^k$ random colorings, the probability of a wrong answer is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

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



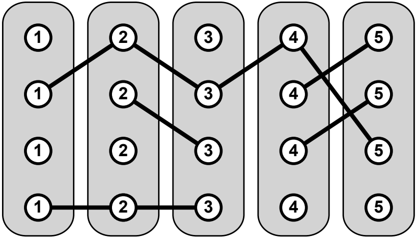
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

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



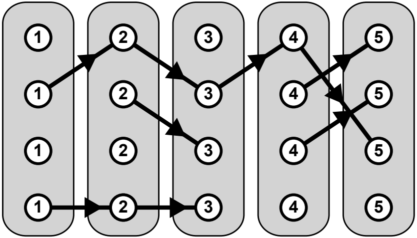
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

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



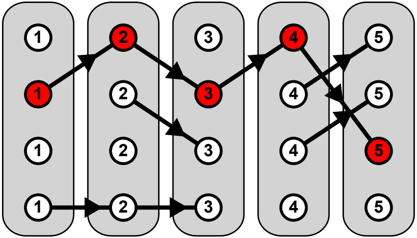
- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

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 $1 - 2 - \cdots - k$

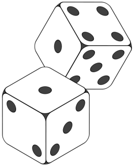


- Edges connecting nonadjacent color classes are removed.
- The remaining edges are directed towards the larger class.
- All we need to check if there is a directed path from class 1 to class k.

Color Coding

k-PATH

Color Coding
success probability:
k^{-k}

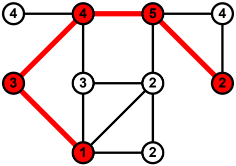


Finding a
$1 - 2 - \cdots - k$
colored path

polynomial-time
solvable

Improved Color Coding

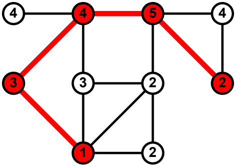
- Assign colors from $[k]$ to vertices $V(G)$ uniformly and independently at random.



- 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.



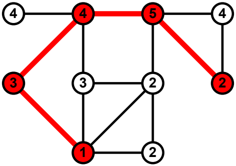
- Check if there is a **colorful** path where each color appears exactly once on the vertices; output "YES" or "NO".
  - If there is no k-path: no **colorful** path exists $\Rightarrow$ "NO".
  - If there is a k-path: the probability that it is **colorful** is

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

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.

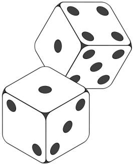
Initialization & Recurrence:

Exercise.

Improved Color Coding

k-PATH

Color Coding
success probability:
e^{-k}



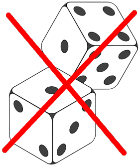
Finding a colorful path

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

Derandomized Color Coding

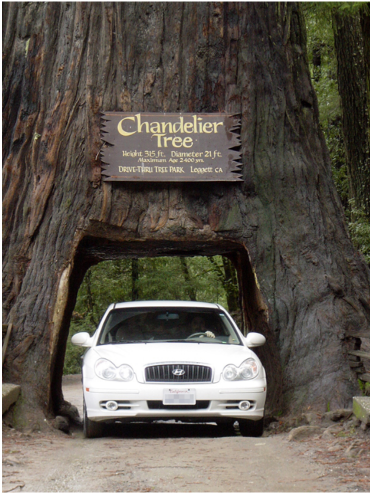
k-PATH

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



Finding a colorful path

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


Chandelier
Tree
Height 315 ft. Diameter 21 ft.
Maximum Age 2400 yrs.
DRIVE-THRU TREE PARK Leggett CA

Treewidth

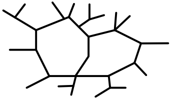
Generalizing trees

How could we define that a graph is “treelike”?

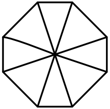
Generalizing trees

How could we define that a graph is "treelike"?

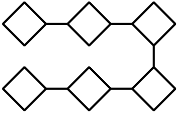
1. Number of cycles is bounded.



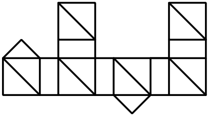
good



bad



bad

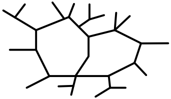


bad

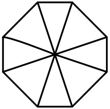
Generalizing trees

How could we define that a graph is "treelike"?

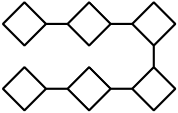
1. Number of cycles is bounded.



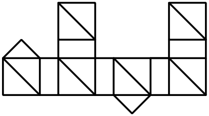
good



bad

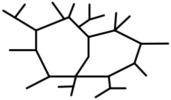


bad

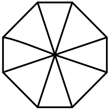


bad

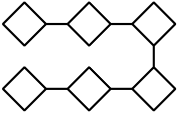
2. Removing a bounded number of vertices makes it acyclic.



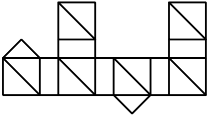
good



good



bad

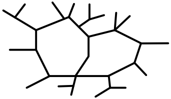


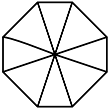
bad

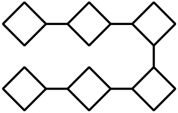
Generalizing trees

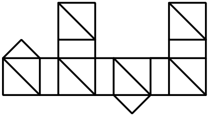
How could we define that a graph is "treelike"?

1. Number of cycles is bounded.

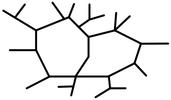

good

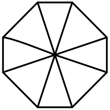

bad

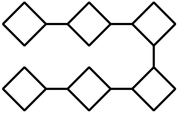

bad

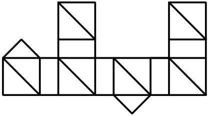

bad

2. Removing a bounded number of vertices makes it acyclic.

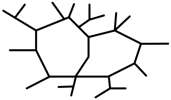

good

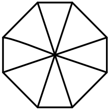

good

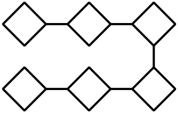

bad

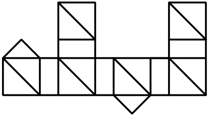

bad

3. Bounded-size parts connected in a tree-like way.


bad


bad

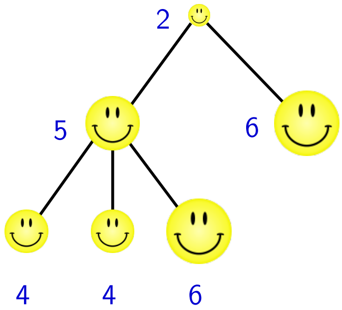

good


good

The Party Problem

PARTY PROBLEM

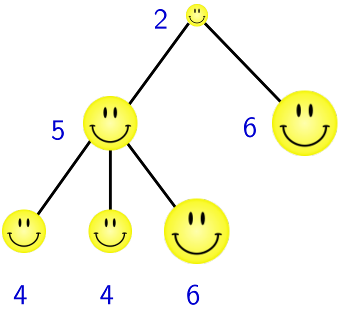
Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.



The Party Problem

Party Problem

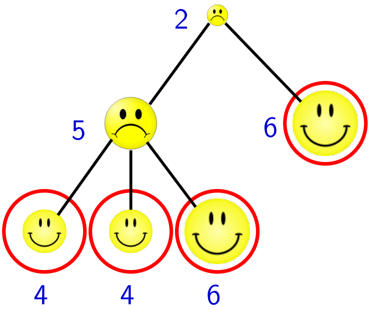
Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	Do not invite a colleague and his direct boss at the same time!



The Party Problem

Party Problem

Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	Do not invite a colleague and his direct boss at the same time!

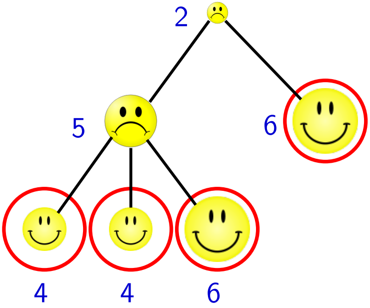


- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

The Party Problem

Party Problem

Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	Do not invite a colleague and his direct boss at the same time!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

Solving the Party Problem

Dynamic programming paradigm:
We solve a large number of subproblems that depend on each other. The answer is a single subproblem.

Subproblems:

T_v: the subtree rooted at v.
$A[v]$: max. weight of an independent set in T_v
$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Solving the Party Problem

Subproblems:

T_v: the subtree rooted at v.

$A[v]$: max. weight of an independent set in T_v

$B[v]$: max. weight of an independent set in T_v that does not contain v

Recurrence:

Assume $v_1, \ldots, v_k$ are the children of v. Use the recurrence relations

$$\boxed{\begin{array}{l} B[v] = \sum_{i=1}^{k} A[v_i] \\ A[v] = \max\{B[v]\ ,\ w(v) + \sum_{i=1}^{k} B[v_i]\} \end{array}}$$

The values $A[v]$ and $B[v]$ can be calculated in a bottom-up order (the leaves are trivial).

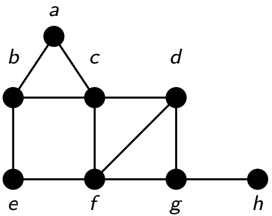
Treewidth — a measure of "tree-likeness"

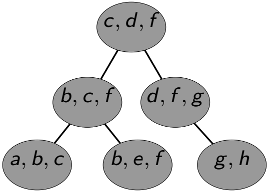
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every v, the bags containing v form a connected subtree.

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.





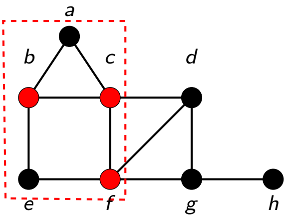
Treewidth — a measure of "tree-likeness"

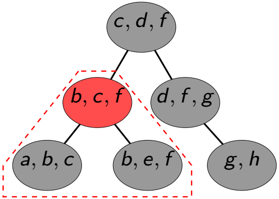
Tree decomposition: Vertices are arranged in a tree structure satisfying the following properties:

1. If u and v are neighbors, then there is a bag containing both of them.
2. For every v, the bags containing v form a connected subtree.

Width of the decomposition: largest bag size -1.

treewidth: width of the best decomposition.





A subtree communicates with the outside world only via the root of the subtree.

Weighted Max Independent Set and treewidth

Theorem

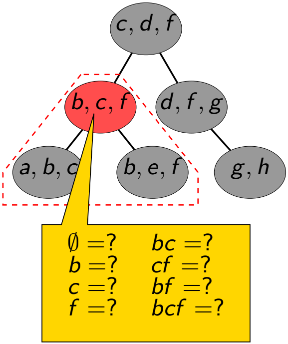
Given a tree decomposition of width w, Weighted Max Independent Set can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

B_x: vertices appearing in node x.
V_x: vertices appearing in the subtree rooted at x.

Generalizing our solution for trees:

Instead of computing 2 values $A[v]$, $B[v]$ for each vertex of the tree, we compute $2^{|B_x|} \leq 2^{w+1}$ values for each bag B_x.

$M[x, S]$:
the max. weight of an independent set $I \subseteq V_x$ with $I \cap B_x = S$.



Weighted Max Independent Set and treewidth

Theorem

Given a tree decomposition of width w, Weighted Max Independent Set can be solved in time $O(2^w \cdot w^{O(1)} \cdot n)$.

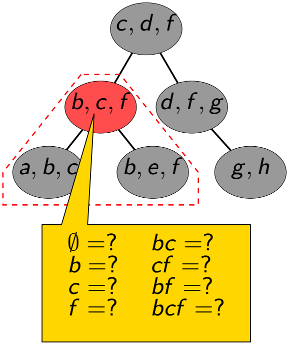
B_x: vertices appearing in node x.
V_x: vertices appearing in the subtree rooted at x.

Generalizing our solution for trees:

Instead of computing 2 values $A[v]$, $B[v]$ for each vertex of the tree, we compute $2^{|B_x|} \leq 2^{w+1}$ values for each bag B_x.

$M[x, S]$:
the max. weight of an independent set $I \subseteq V_x$ with $I \cap B_x = S$.

How to determine $M[x, S]$ if all the values are known for the children of x?



3-Coloring and tree decompositions

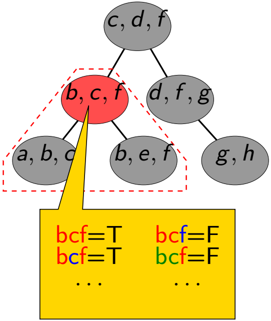
Theorem

Given a tree decomposition of width w, 3-Coloring can be solved in time $3^w \cdot w^{O(1)} \cdot n$.

B_x: vertices appearing in node x.
V_x: vertices appearing in the subtree rooted at x.

For every node x and coloring $c : B_x \to \{1, 2, 3\}$, we compute the Boolean value $E[x, c]$, which is true if and only if c can be extended to a proper 3-coloring of V_x.

Claim:
We can determine $E[x, c]$ if all the values are known for the children of x.



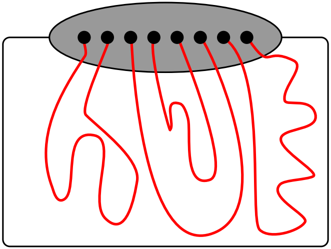
Tree decompositions and dynamic programming

General scheme: Define subproblems for each subtree and solve them in a bottom up manner.

Number of subproblems:

- 3-Coloring: 3^{w+1}
  (number of 3-colorings of the bag)
- Independent Set: 2^{w+1}
  (each vertex of the bag is either in the solution or not)
- Dominating Set: 3^{w+1}
  (each vertex of the bag is either (1) in the solution, (2) not in the solution, but dominated, (3) not in the solution and not yet dominated)
- Hamiltonian Cycle: $w^{O(w)} = 2^{O(w \log w)}$
  (number of ways the paths of the partial solution can match vertices of the bag).

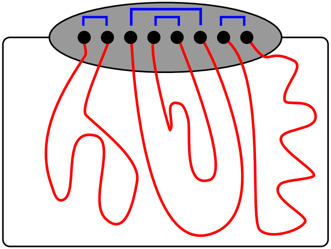
Number of subproblems for Hamiltonian Cycle



To describe a partial solution, we need to describe the matching of the bag formed by the paths in the partial solution.

Number of matchings: $w^{O(w)} \Rightarrow$ the textbook dynamic programming algorithm has running time $w^{O(w)} \cdot n^{O(1)}$.

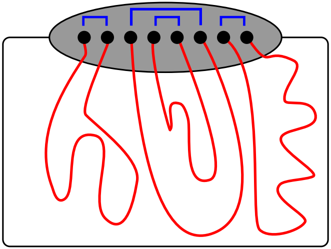
Number of subproblems for HAMILTONIAN CYCLE



To describe a partial solution, we need to describe the matching of the bag formed by the paths in the partial solution.

Number of matchings: $w^{O(w)} \Rightarrow$ the textbook dynamic programming algorithm has running time $w^{O(w)} \cdot n^{O(1)}$.

Number of subproblems for HAMILTONIAN CYCLE



To describe a partial solution, we need to describe the matching of the bag formed by the paths in the partial solution.

Number of matchings: $w^{O(w)} \Rightarrow$ the textbook dynamic programming algorithm has running time $w^{O(w)} \cdot n^{O(1)}$.

But, surprisingly, it is possible to solve HAMILTONIAN CYCLE in time $2^{O(w)} \cdot n^{O(1)}$!

Cut and count

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

Classical result:

Theorem [textbook algorithm]

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

Improved algorithm:

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

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

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

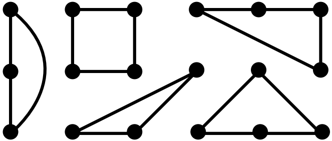
$$1 - \frac{|U|}{N}.$$

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.

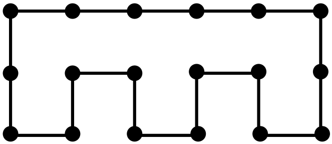
Cycle covers

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



Cycle covers

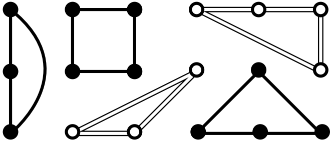
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



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

Cycle covers

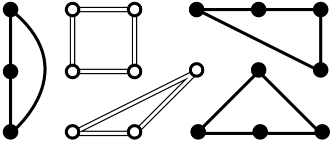
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- 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

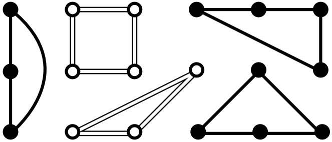
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- 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

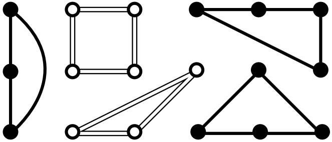
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- 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.

Cycle covers

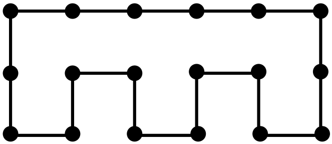
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- 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 $0 \bmod 4$.
  - If there is a unique weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is $2 \bmod 4$.

Cycle covers

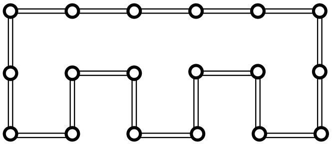
- **Cycle cover:** A subgraph having degree exactly two at each vertex.



- 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 $0 \bmod 4$.
  - If there is a unique weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is $2 \bmod 4$.

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 $0 \bmod 4$.
  - If there is a unique weight-C Hamiltonian cycle: the number of weight-C colored cycle covers is $2 \bmod 4$.

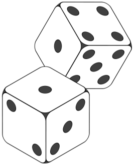
Cut and Count

- 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 $4^w \cdot n^{O(1)}$ if a tree decomposition of width w is given.
- Answer YES if this number is $2 \bmod 4$.

Cut and Count

HAMILTONIAN CYCLE

Random weights
success probability:
1/2



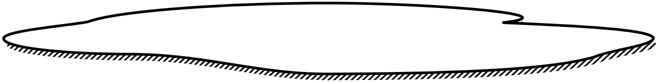
Counting weighted colored cycle covers

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

Treewidth

There are two ways in which we can encounter bounded-treewidth graphs:

1. Designing algorithms for graphs of bounded treewidth.
    - Which problems can be solved efficiently on such graphs?
    - What is the best possible dependence of the running time on treewidth?
2. Using bounded-treewidth algorithms as subroutines.
    - Most notably for planar graphs.



Planar graphs

Subexponential algorithm for 3-Coloring

Theorem [textbook dynamic programming]

3-Coloring can be solved in time $2^{O(w)} \cdot n^{O(1)}$ on graphs of treewidth w.

$+$

Theorem [Robertson and Seymour]

A planar graph on n vertices has treewidth $O(\sqrt{n})$.

Subexponential algorithm for 3-Coloring

Theorem [textbook dynamic programming]

3-Coloring can be solved in time $2^{O(w)} \cdot n^{O(1)}$ on graphs of treewidth w.

$+$

Theorem [Robertson and Seymour]

A planar graph on n vertices has treewidth $O(\sqrt{n})$.

$\Downarrow$

Corollary

3-Coloring can be solved in time $2^{O(\sqrt{n})}$ on planar graphs.

textbook algorithm + combinatorial bound

$\Downarrow$

subexponential algorithm

Subexponential planar algorithms using treewidth

We need only the following basic facts:

Treewidth

1. If a graph G has treewidth w, then many classical NP-hard problems can be solved in time $2^{O(w)} \cdot n^{O(1)}$ or $2^{O(w \log w)} \cdot n^{O(1)}$ on G.
2. A planar graph on n vertices has treewidth $O(\sqrt{n})$.

This immediately gives subexponential-time ($2^{O(\sqrt{n})}$ or $2^{O(\sqrt{n} \log n)}$) algorithms for many problems on planar graphs.

- 3-Coloring
- Hamiltonian Cycle
- Independent Set
- Vertex Cover
- ...

Subexponential planar algorithms using treewidth

We need only the following basic facts:

Treewidth

1. If a graph G has treewidth w, then many classical NP-hard problems can be solved in time $2^{O(w)} \cdot n^{O(1)}$ or $2^{O(w \log w)} \cdot n^{O(1)}$ on G.
2. A planar graph on n vertices has treewidth $O(\sqrt{n})$.

Next:

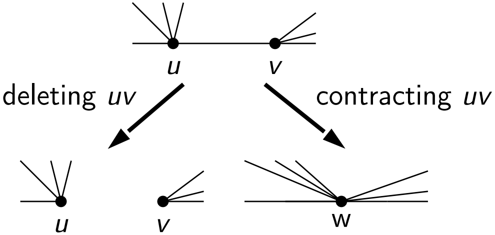
What about parameterized problems? Can we make $f(k)$ subexponential for Vertex Cover or k-Path on planar graphs?

But first, let's see the reason why an n-vertex planar graph has treewidth $O(\sqrt{n})$.

Minors

Definition

Graph H is a **minor** of G ($H \leq G$) if H can be obtained from G by deleting edges, deleting vertices, and contracting edges.

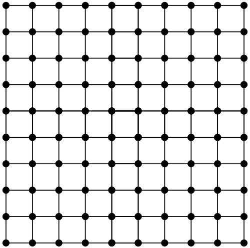


Note: length of the longest path in H is at most the length of the longest path in G.

Planar Excluded Grid Theorem

Theorem [Robertson, Seymour, Thomas 1994]

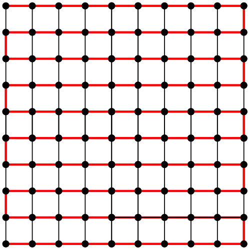
Every planar graph with treewidth at least $5k$ has a $k \times k$ grid minor.



Note: for general graphs, treewidth at least k^{100} or so guarantees a $k \times k$ grid minor [Chekuri and Chuzhoy 2013]!

Bidimensionality for k-Path

Observation: If the treewidth of a planar graph G is at least $5\sqrt{k}$
$\Rightarrow$ It has a $\sqrt{k} \times \sqrt{k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a path of length at least k.
$\Rightarrow$ G has a path of length at least k.

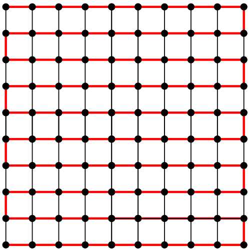


Bidimensionality for k-PATH

Observation: If the treewidth of a planar graph G is at least $5\sqrt{k}$
$\Rightarrow$ It has a $\sqrt{k} \times \sqrt{k}$ grid minor (Planar Excluded Grid Theorem)
$\Rightarrow$ The grid has a path of length at least k.
$\Rightarrow$ G has a path of length at least k.

We use this observation to find a path of length at least k on planar graphs:

- Set $w := 5\sqrt{k}$.
- Find an $O(1)$-approximate tree decomposition.
  - If treewidth is at least w: we answer "there is a path of length at least k."
  - If we get a tree decomposition of width $O(w)$, then we can solve the problem in time $2^{O(w \log w)} \cdot n^{O(1)} = 2^{O(\sqrt{k} \log k)} \cdot n^{O(1)}$.

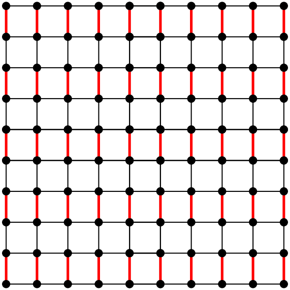


Bidimensionality

Definition

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \le x(G)$ for every minor G' of G, and
- If G_k is the $k \times k$ grid, then $x(G_k) \ge ck^2$ (for some constant $c > 0$).



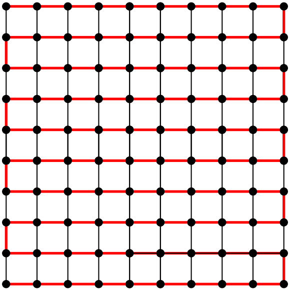
Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

Bidimensionality

Definition

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor G' of G, and
- If G_k is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).



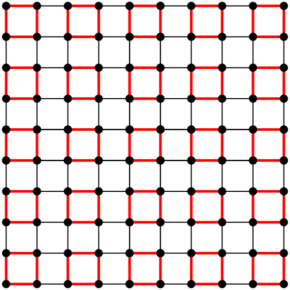
Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

Bidimensionality

Definition

A graph invariant $x(G)$ is **minor-bidimensional** if

- $x(G') \leq x(G)$ for every minor G' of G, and
- If G_k is the $k \times k$ grid, then $x(G_k) \geq ck^2$ (for some constant $c > 0$).



Examples: minimum vertex cover, length of the longest path, feedback vertex set are minor-bidimensional.

Square root phenomenon for planar graphs

- Simple $2^{O(\sqrt{n})}$ time algorithms for planar graphs by using that planar graphs have treewidth $O(\sqrt{n})$.
- Simple $2^{O(\sqrt{k})} \cdot n^{O(1)}$ time parameterized algoritms using bidimensionality.
- More complicated and problem-specific algorithms for problems where bidimentsionality does not work (Steiner Tree, Subset TSP).
- $n^{O(\sqrt{k})}$ time algorithms for W[1]-hard problems.

In many cases, these algorithms are optimal. More about this on Thursday at 3pm...

Wrap up

- The FPT vs. W[1]-hard game
- The $f(k)$ game for FPT problems
- The exponent game for W[1]-hard problems

We have seen that many nontrivial positive results were obtained for these questions.

Next: what about negative results?