Fun with Parameterized Complexity

Fernando Sánchez Villaamil

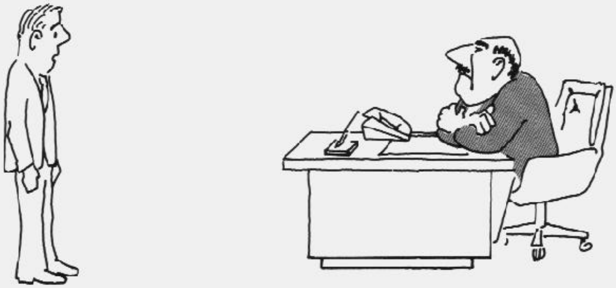
Felix Reidl

Theoretical Computer Science

RWTH AACHEN

@NCSU 2014

NP-hardness:
where dreams go to die



"I can't find an efficient algorithm, I guess I'm just too dumb."

NP-hardness:
where dreams go to die

"I can't find an efficient algorithm, but neither can all these famous people."

NP-hardness:
where dreams go to die

About ten years ago, some computer scientists came by and said they heard we have some really cool problems. They showed that the problems are NP-complete and went away!
—Joseph Felsenstein (Molecular biologist)

Felix Reidl & Fernando Sánchez Villaamil

What does NP-hard mean?

The problem is *hard*.

Colors!

4-COLORABILITY

Input:	A graph G
Problem:	Can the vertices of G be colored with 4 colors, such that no edge is monochromatic?

NP-complete

PLANAR 4-COLORABILITY

Input:	A *planar* graph G
Problem:	Can the vertices of G be colored with 4 colors, such that no edge is monochromatic?

Constant time

What does NP-hard mean?

The problem is *hard*
on *some* instances.

Clique

k-CLIQUE
Input: A graph G
Problem: Does G contains a complete subgraph on k vertices?

Definition (3-Clique)

Does G contains a complete subgraph on 3 vertices?

Can be solved in $O(n^3)$.

Definition (10000-Clique)

Does G contains a complete subgraph on 10000 vertices?

Can be solved in $O(n^{10000})$.

What does NP-hard mean?

The problem is *hard* on *some* instances.*

***If the parameter is unbounded.**

k-COLORABILITY

k-COLORABILITY

Input: A graph G

Problem: Can the vertices of G be colored with k colors, such that no edge is monochromatic?

Can k-COLORABILITY be solved in time $O(n^k)$ on general graphs?

This would imply $\mathbf{P} = \mathbf{NP}$.

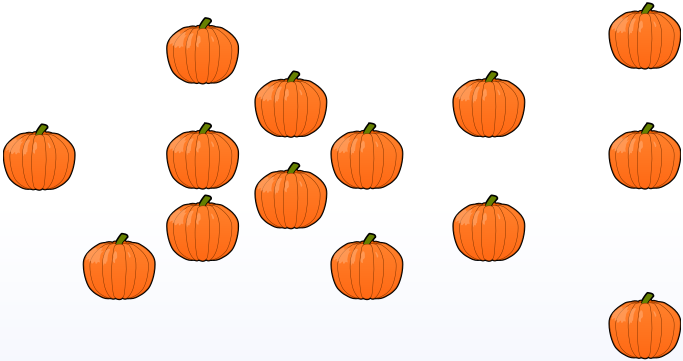
What does NP-hard mean?

The problem is *hard* on *some* instances.*

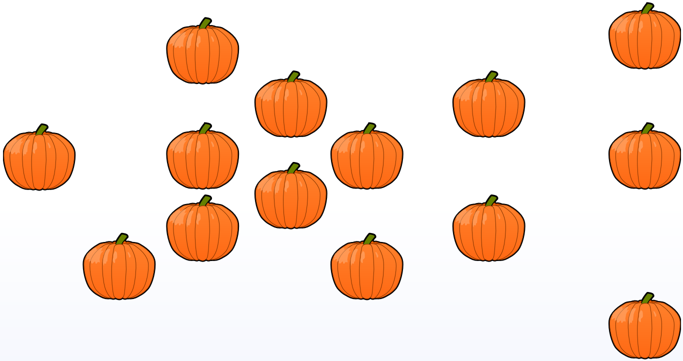
***Sometimes it's only hard if the parameter is unbounded.**

Sometimes it remains hard irrespective of the parameter size.

A seasonal problem

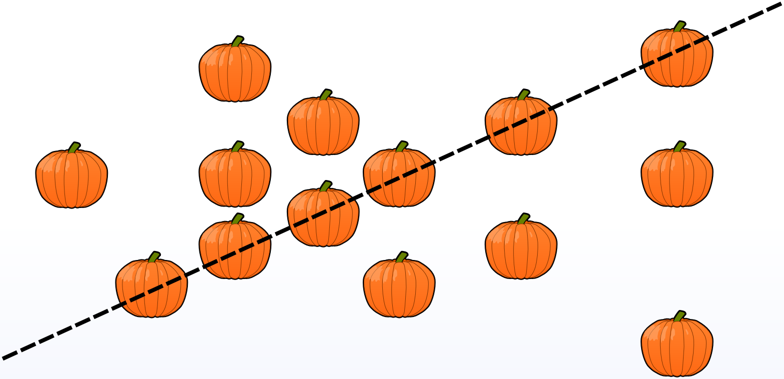


A seasonal problem



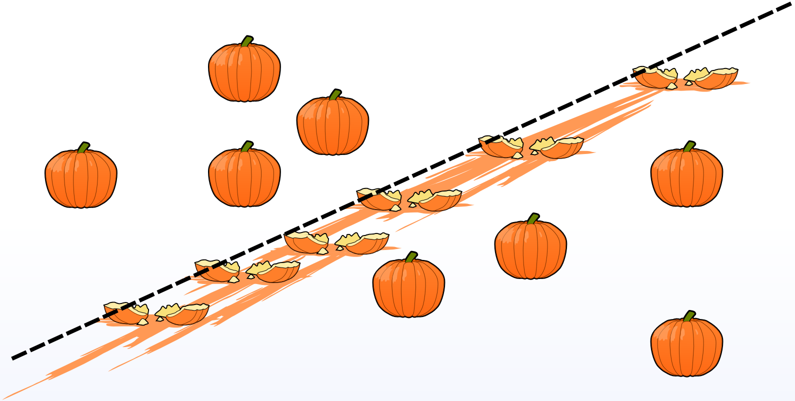
Can you shoot your neighbor’s pumpkins with four shots?

A seasonal problem



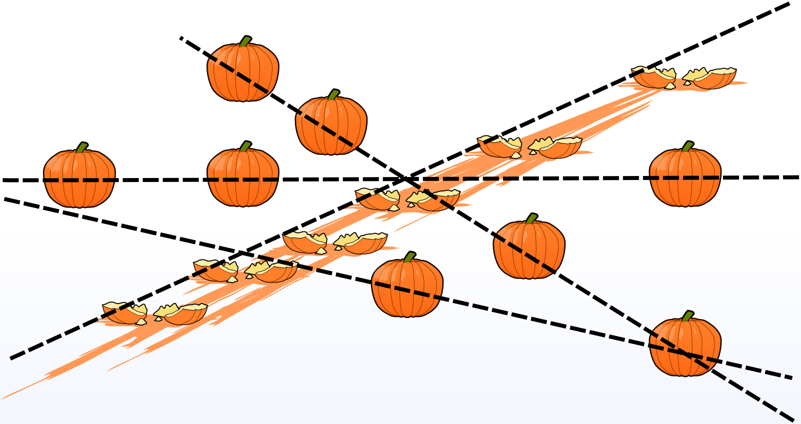
Can you shoot your neighbor’s pumpkins with four shots?

A seasonal problem



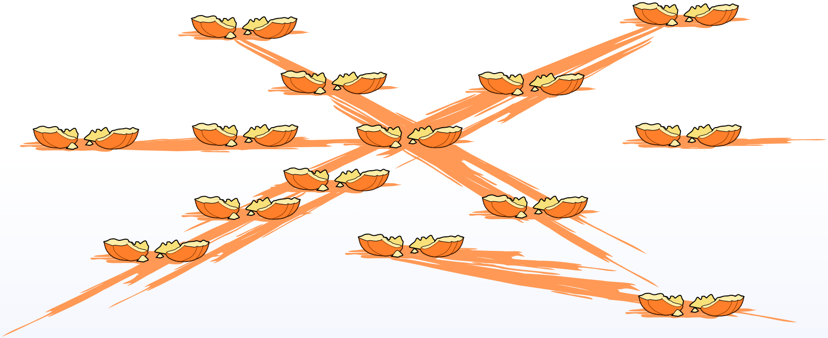
Can you shoot your neighbor's pumpkins with four shots?

A seasonal problem



Can you shoot your neighbor's pumpkins with four shots?

A seasonal problem



Can you shoot your neighbor's pumpkins with four shots? Yes, and in time $O(k^{2k} \cdot n)$

What does NP-hard mean?

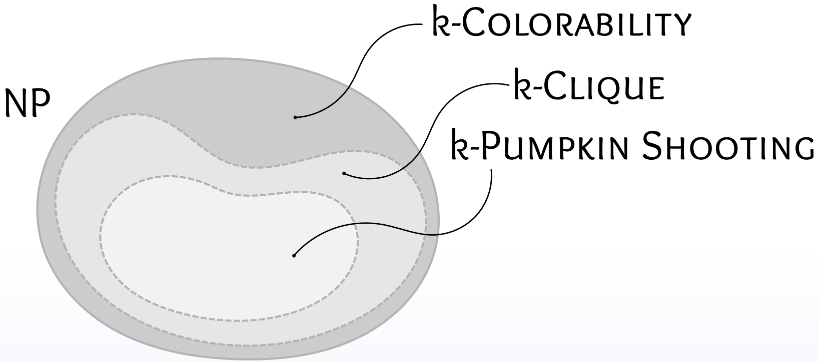
The problem is *hard* on *some* instances.*

***Sometimes it's only hard if the parameter is unbounded.**

Sometimes it remains hard irrespective of the parameter size.

↳ *Sometimes we can decouple the complexity of the input size and the parameter, other times this does not seem to work...*

Fine structure of NP



The magical k lets us distinguish between these problems

We call it the parameter

Definitions!

Definition (Parameter)

A *parameter* is given by a polynomial-time computable function, which maps instances of our problem to natural numbers.

Definition ($\mathbf{XP}$)

A problem is in $\mathbf{XP}$ parameterized by k if there exists an algorithm which solves the problem in time $O(n^{f(k)})$.

Definition ($\mathbf{FPT}$)

A problem is *fixed parameter tractable* parameterized by k if there exists an algorithm which solves the problem in time $f(k) \cdot n^{O(1)}$. In this case we say the problem is in $\mathbf{FPT}$.

Vertex Cover

Vertex Cover

Input:	A graph G, an integer k
Problem:	Is there a vertex set $S \subseteq V(G)$ of size at most k such that every edge of G has at least one endpoint in S?

- It is easy to see Vertex Cover is in $\mathbf{XP}$.
- It is also one of the famous problems in $\mathbf{FPT}$...

Interlude: a funny story about VERTEX COVER

Hammer time

Theorem (Robertson & Seymour)

Every minor-closed property is recognizable in time $O(n^3)$ time.

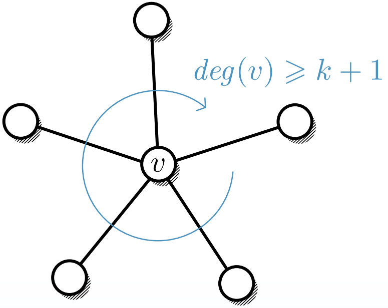
For every fixed k, having a vertex cover of size at most k is a minor closed property.

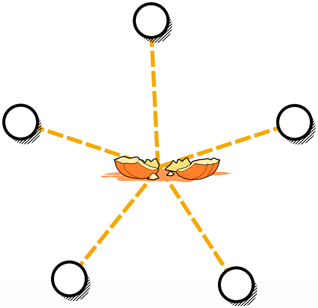
Corollary

Vertex cover is solvable in time $f(k) \cdot n^3$.

> *If the constants in Robertson & Seymour's minors theorem are your friends, you don't need enemies.*
>
> *–Daniel Marx*

A pumpkin-style argument





- Each time we apply the rule, we decrease k by one

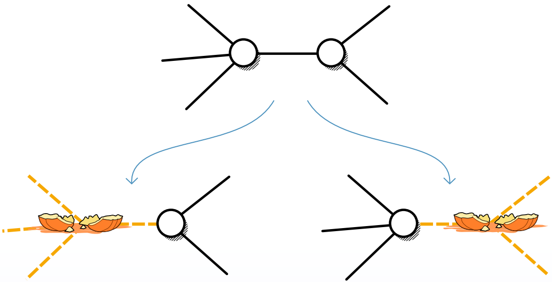
$\Rightarrow$ Can happen at most k times

- At the end, the degree of every vertex is at most k

$\Rightarrow$ The remaining graph has size k^2

Brute-force remainder in time $O(2^{k^2})$

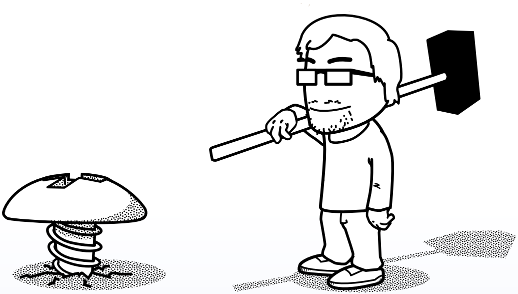
But but but



- We *branch* into two subcases, both with the parameter decreased by one
- If $k = 0$ or no edges left: trivial
- *Search tree* hence is bounded by $O(2^k)$

This solves VERTEX COVER in time $O(2^k \cdot n)$

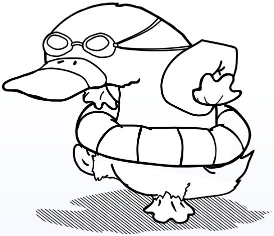
The lesson



If all you have is a hammer,
everything looks like a nail.

Not so rare or complicated

- Initially people thought that few problems would be in **FPT**, that proving it would be complicated and that the algorithms would be complex.
- Luckily, a large number of problems have simple fpt-algorithms: k-VERTEX COVER, k-CONNECTED VERTEX COVER, k-CENTERED STRING, k-TRIANGLE DELETION, k-CLUSTER EDITING, k-MAX LEAF SPANNING TREE, k-3-HITTING SET...



ML-type languages

What is the complexity of compiling OCaml, Haskell and Scala?

It is EXPTIME-complete!

Yet we compile them?

There exists an algorithm to compile ML-type languages that runs in time $O(2^k \cdot n)$, where k is the nesting-depth of type declarations.

Implication: For any fixed k there exists a compiler that can compile an ML-type language with maximal type nesting-depth of k in *linear time*.

fpt-algorithms in practice

- ML-languages compilation
- Database queries
- Computing evolutionary trees based on binary character information
- Generating a maximum agreement tree from several evolutionary trees
- Parallelization problems parameterized by the number of processors
- Enumeration problems in complex networks
  (our cooperation with Dr. Sullivan)

Furthermore, the design of fpt-algorithms is a great guideline for possible heuristics.

FPT Meta-problems and theorems

- Graph isomorphism parameterized by treewidth
- Deleting k vertices in a graph to make it have any hereditary property
- ILP parameterized by the number of variables
- FO-model checking on nowhere-dense graphs, parameterized by formula size
- EMSO-model checking parameterized by treewidth
- MSO_1-model checking parameterized by rank-width

Why did it take so long?

I think the algorithmic landscape at that time was relatively complacent. Most problems of interest had already been found either to reside in P or to be NP-complete. Thus, natural problems were largely viewed under the classic Jack Edmonds style dichotomy as being good or bad, easy or hard, with not much of a middle ground.

—Michael Langston

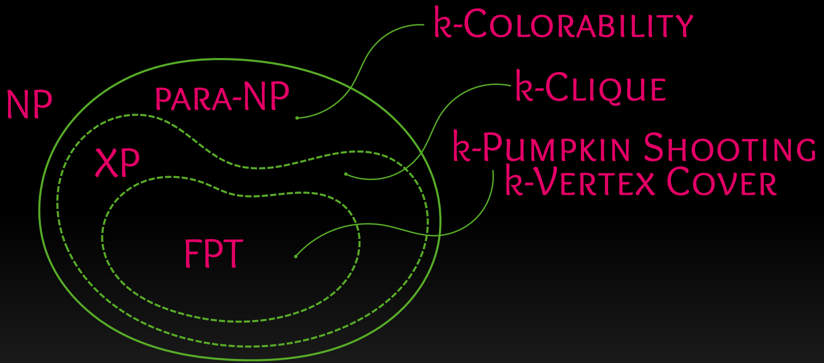
The negative side: intractability

- A positive toolkit is great, but we also want to know when parameterization cannot help
- So, why is k-CLIQUE apparently not fpt? As so often, we only have relative answers...
- Hierarchy: $\mathbf{FPT} \subset W[1] \subset W[2] \subset \ldots$

We strongly believe that $\mathbf{FPT} \neq W[1]$

- k-CLIQUE is $W[1]$-complete
- k-INDEPENDENT SET is $W[1]$-complete
- k-DOMINATING SET is $W[2]$-complete

Fine structure of NP, named



$\mathbf{XP} \neq \mathbf{NP}$ unless $\mathbf{P} = \mathbf{NP}$

we believe that $\mathbf{FPT} \neq \mathbf{XP}$

Parameters, revisited

If a problem is not in $\mathbf{FPT}$ or the natural parameter is just too large, do not give up!

There are a lot of alternative parameters:

- **Structural parameters**: treewidth, rank-width, vertex cover size, feedback vertex set number, degeneracy, distance to triviality,...
- **Improvement parameters**: local-search distance, above-guarantee, reoptimization,...
- **Other**: approximation quality, any combination of the above

Also, parameterized algorithms work very well on *sparse instances*!

Why parameterized complexity?

INDEPENDENT SET
is NP-hard

fpt-approximation scheme
for bounded genus

additive
fpt-approximation
is W[1] hard

fpt parameterized
above guarantee on
planar graphs

INDEPENDENT SET
is W[1]-hard

local search ftp
parameterized by
exchange size

fpt on nowhere
dense classes

$2^{\sqrt{k}} poly(n)$ on
planar graphs

$2^{tw} poly(n)$ parameterized
by treewidth

fpt on
triangle-free graphs

fpt parameterized
by vertex cover

no $(2-\epsilon)^{tw} poly(n)$
algorithm

fpt on degenerate
graphs

fpt on
bull-free graphs

fpt parameterized
by fvs

fpt on
K_r-free graphs

fpt parameterized
by P_3 cover

Parameterized algorithms for the unconvinced

Preprocessing

A preprocessing algorithm takes an instance of a (hard) problem and outputs an equivalent, smaller instance.

Preprocessing is used everywhere in practice and seems to work amazingly well!

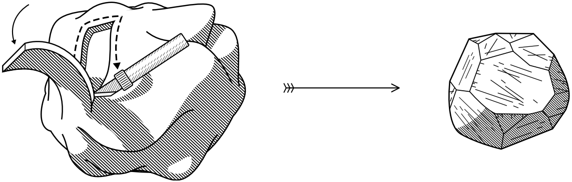
Question: Can there exist a preprocessing algorithm for an $\mathbf{NP}$-hard problem of size n that produces an instance of size ϵn for some $\epsilon < 1$?

Not unless $\mathbf{P} = \mathbf{NP}$!

No formal analysis of preprocessing possible?

- We need some measure that tells us when we cannot preprocess an instance further!
- Very natural for **parameterized problems**!
- Actually, we already saw two examples:
  - k-PUMPKIN SHOOTING
  - k-VERTEX COVER
- Basic idea: perform basic reduction rules exhaustively, then use remaining structure to prove that instance is small

Kernelization



Definition (Kernelization)

A kernelization for a parameterized problem L is an algorithm that takes an instance (x, k) and maps it in time polynomial in $|x|$ and k to an instance (x', k') such that

- $(x, k) \in L \Leftrightarrow (x', k') \in L$,
- $k' + |x'| \leq f(k)$

where f is a function we call the *size* of the kernel.

Does not contradict $\mathbf{P} \neq \mathbf{NP}$

$\mathbf{FPT}$ is the same as kernelization

Proof.

Assume you have an algorithm that solves a problem parameterized by k in time $f(k) \cdot n^c$. Then we have an $f(k)$ kernelization algorithm:

- If $n \leq f(k)$ don't do anything.
- Else $f(k) \cdot n^c < n^{c+1}$, thus our algorithm has polynomial running time. Solve the instance and output a trivial YES or NO instance.

□

Polynomial kernels

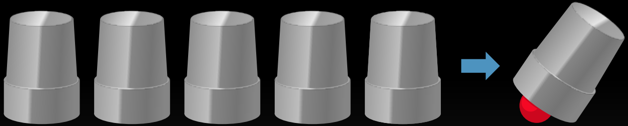
Astoundingly, we can for some problems—in polynomial time—compute an equivalence instance of size polynomial in k.

- Gives you instances where even brute-force might be reasonable
- We have seen two examples: k-PUMPKIN SHOOTING and k-VERTEX COVER
- Other examples are: k-FEEDBACK VERTEX SET, k-PLANAR DOMINATING SET, k-CLUSTER VERTEX DELETION and many problems when restricted to sparse graphs.

Do all problems in $\mathbf{FPT}$ have a poly-kernel?

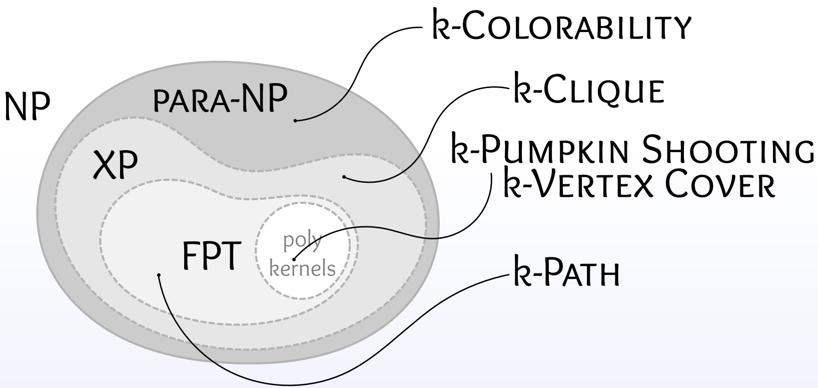
Problems without a poly-kernel

For some problems it seemed unlikely to find a poly-kernel, e.g. k-PATH:



In 2008 Bodlaender, Downey, Fellows and Hermelin provided a framework to prove that many problems do not have a poly-kernel under the assumption that $\mathbf{NP} \not\subseteq \mathbf{coNP/poly}$.
(has subsequently been refined and improved)

The parameterized landscape inside NP



Takeaway

- The $\mathbf{NP}$ vs. $\mathbf{P}$ framework alone is insufficient to understand complexity of important problems.
- $\mathbf{FPT}$ provides a rich and satisfying framework for multivariate complexity analysis—both on the positive and negative side.
- Besides approximation, ILPs and heuristic one should be aware of **fpt algorithms**!

If you think it might be applicable to some of your problems, drop us a line.

Thank you!