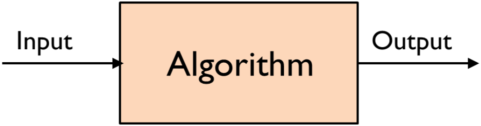
Towards a Tight Understanding of the Complexity of Algorithmic Problems

Dániel Marx

Max Planck Institute for Informatics
Saarbrücken, Germany

January 8, 2020

Theory of Algorithms



- Worst-case analysis: guaranteed running time for every input of size n.
- Two main classes:
  - Polynomial time ($O(n)$, $O(n \log n)$, $O(n^2)$, $\ldots$)
  - Exponential time (2^n, $2^{\sqrt{n}}$, $\ldots$)

Rule of theory

Classical theory focuses on polynomial-time:

Theory of algorithms
Solve problems in
polynomial time

Computational complexity
Use NP-completeness
for negative evidence

Rule of theory

Classical theory focuses on polynomial-time:

~~**Theory of algorithms**~~
~~Solve problems in polynomial time~~

~~**Computational complexity**~~
~~Use NP-completeness for negative evidence~~

But this is only a restricted view of the picture:

Theory of algorithms
Give nontrivial insight into the problem

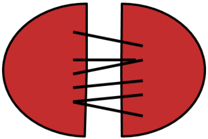
Computational complexity
Show that the current best algorithms are optimal

We want a tight understanding of all the ideas relevant to a particular problem.

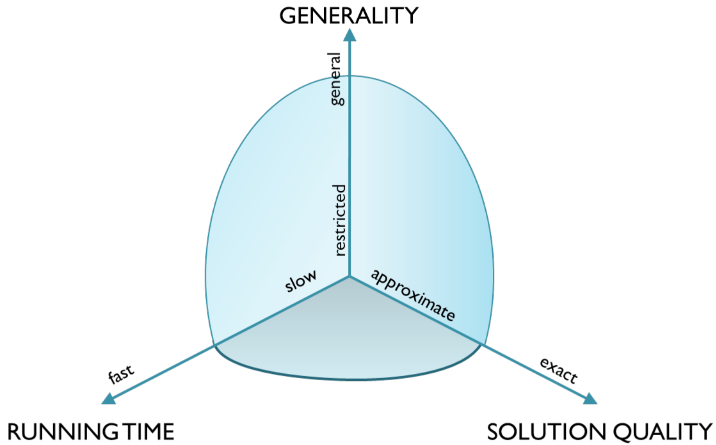
A classic tight result

Tight result on the approximability of MAX CUT:

- Polynomial-time 0.878-approximation using semidefinite programming (SDP) on general graphs.
  [Goemans and Williamson 1994]
- Complexity-theoretic evidence that no polynomial-time approximation on general graphs with ratio $0.878 + \epsilon$.
  [Khot et al. 2004]



Dimensions



GENERALITY
general
restricted
slow
approximate
fast
exact
RUNNING TIME
SOLUTION QUALITY

Dimensions

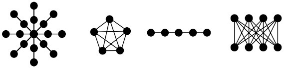
- **Running time**
  - Polynomial $\leftrightarrow$ exponential

$O(n)$ $O(n^2)$ $n^{O(1)}$ $n^{O(\log n)}$ $2^{O(\sqrt{n})}$ $2^{n^{O(1)}}$ 2^{2^n}

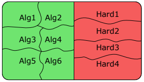
  - Optimality program in parameterized complexity

$$f(k)n^{O(1)} \leftrightarrow n^{O(k)}$$

- **Generality**
  - Study of special cases



  - Complete classification results



- **Solution quality**
  - Approximation, PTASs
  - Parameterized approximation

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

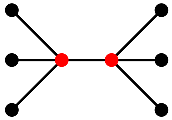
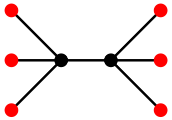
Instead of expressing the running time as a function $T(n)$ of n, we express it as a function $T(n, k)$ of the input size n and some parameter k of the input.

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?

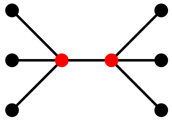
- The size k of the solution we are looking for.
- The maximum degree of the input graph.
- The dimension of the point set in the input.
- The length of the strings in the input.
- The length of clauses in the input Boolean formula.
- $\dots$

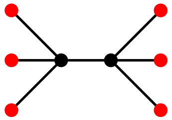
Parameterized complexity

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

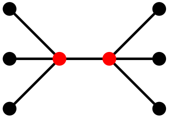
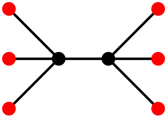
Problem:	VERTEX COVER	INDEPENDENT SET
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?





Complexity:	NP-complete	NP-complete
Brute force:	$O(n^k)$ possibilities	$O(n^k)$ possibilities

Parameterized complexity

	Vertex Cover	Independent Set
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**	**No** $n^{o(k)}$ **algorithm known**

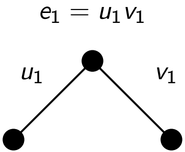
Bounded search tree method

Algorithm for VERTEX COVER:

$e_1 = u_1 v_1$

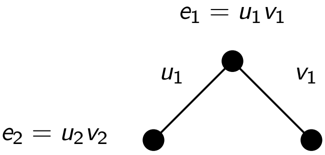
Bounded search tree method

Algorithm for Vertex Cover:



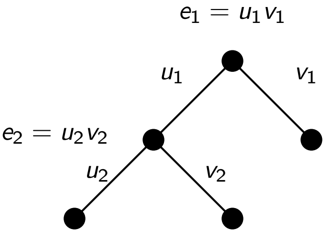
Bounded search tree method

Algorithm for Vertex Cover:



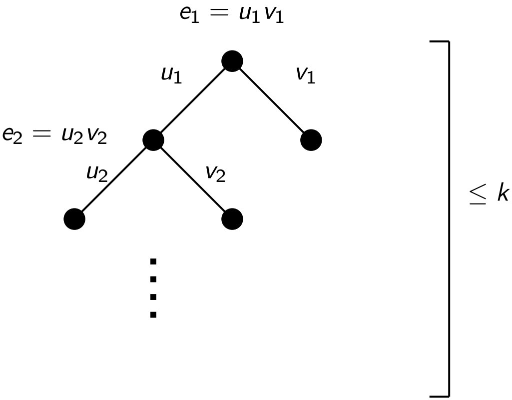
Bounded search tree method

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

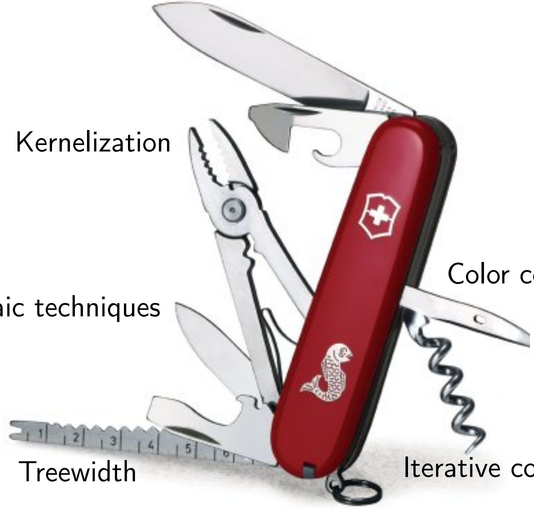
Main definition

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



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:

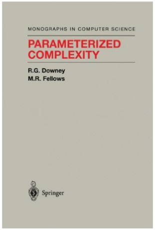
- Finding a clique/independent set of size k.
- Finding a dominating set of size k.
- Finding k pairwise disjoint sets.
- ...

Parameterized complexity

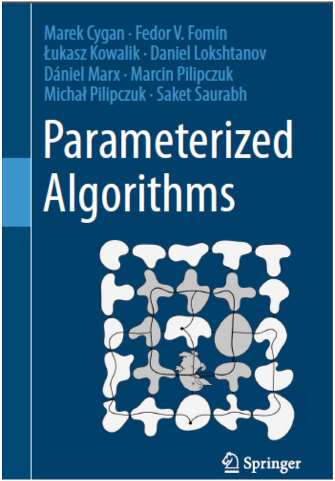
Rod G. Downey
Michael R. Fellows

Parameterized Complexity

Springer 1999



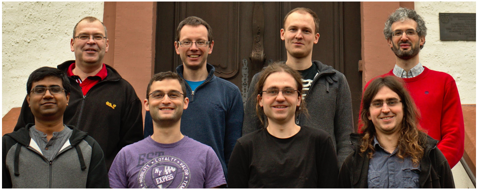
- The study of parameterized complexity was initiated by Downey and Fellows in the early 90s.
- First monograph in 1999.
- By now, strong presence in most algorithmic conferences.



Parameterized Algorithms

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

Springer 2015



Shift of focus

qualitative question

FPT or W[1]-hard?

Shift of focus

qualitative question

FPT or W[1]-hard?

FPT

W[1]-hard

quantitative question

What is the best possible multiplier $f(k)$ in the running time $f(k) \cdot n^{O(1)}$?

2^k? 1.0001^k? $2^{\sqrt{k}}$?

What is the best possible exponent $g(k)$ in the running time $f(k) \cdot n^{g(k)}$?

$n^{O(k)}$? $n^{\log k}$? $n^{\log \log k}$?

Better algorithms for VERTEX COVER

- We have seen a $2^k \cdot n^{O(1)}$ time algorithm.
- Easy to improve to, e.g., $1.618^k \cdot n^{O(1)}$.
- Current best $f(k)$: $1.2738^k \cdot n^{O(1)}$ [Chen, Kanj, Xia 2010].
- Lower bounds?
  - Is, say, $1.001^k \cdot n^{O(1)}$ time possible?
  - Is $2^{k/\log k} \cdot n^{O(1)}$ time possible?

Better algorithms for Vertex Cover

- We have seen a $2^k \cdot n^{O(1)}$ time algorithm.
- Easy to improve to, e.g., $1.618^k \cdot n^{O(1)}$.
- Current best $f(k)$: $1.2738^k \cdot n^{O(1)}$ [Chen, Kanj, Xia 2010].
- Lower bounds?
  - Is, say, $1.001^k \cdot n^{O(1)}$ time possible?
  - Is $2^{k/\log k} \cdot n^{O(1)}$ time possible?

Of course, for all we know, it is possible that $P = NP$ and Vertex Cover is polynomial-time solvable.

$\Rightarrow$ We can hope only for conditional lower bounds.

Exponential Time Hypothesis (ETH)

Hypothesis introduced by Impagliazzo, Paturi, and Zane:

Exponential Time Hypothesis (ETH) [consequence of]

There is no $2^{o(n)}$-time algorithm for n-variable $\mathrm{3SAT}$.

Note: current best algorithm is 1.30704^n [Hertli 2011].

Note: an n-variable $\mathrm{3SAT}$ formula can have $m = \Omega(n^3)$ clauses.

Exponential Time Hypothesis (ETH)

Hypothesis introduced by Impagliazzo, Paturi, and Zane:

Exponential Time Hypothesis (ETH) [consequence of]

There is no $2^{o(n)}$-time algorithm for n-variable $\mathrm{3SAT}$.

Note: current best algorithm is 1.30704^n [Hertli 2011].

Note: an n-variable $\mathrm{3SAT}$ formula can have $m = \Omega(n^3)$ clauses.

Are there algorithms that are subexponential in the size $n + m$ of the $\mathrm{3SAT}$ formula?

Sparsification Lemma [Impagliazzo, Paturi, Zane 2001]

There is a $2^{o(n)}$-time algorithm for n-variable $\mathrm{3SAT}$.

$$\Updownarrow$$

There is a $2^{o(n+m)}$-time algorithm for n-variable m-clause $\mathrm{3SAT}$.

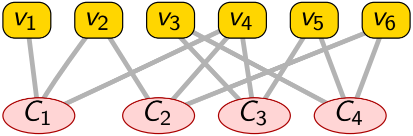
Lower bounds based on ETH

Exponential Time Hypothesis (ETH)

There is no $2^{o(n+m)}$-time algorithm for n-variable m-clause 3SAT.

The textbook reduction from 3SAT to Vertex Cover:

3SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G $O(n+m)$ vertices $O(n+m)$ edges



Corollary

Assuming ETH, there is no $2^{o(n)}$ algorithm for Vertex Cover on an n-vertex graph G.

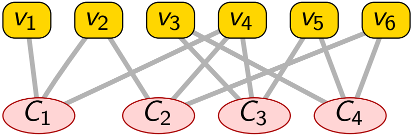
Lower bounds based on ETH

Exponential Time Hypothesis (ETH)

There is no $2^{o(n+m)}$-time algorithm for n-variable m-clause 3SAT.

The textbook reduction from 3SAT to Vertex Cover:

3SAT formula ϕ n variables m clauses	$\Rightarrow$	Graph G $O(n+m)$ vertices $O(n+m)$ edges



Corollary

Assuming ETH, there is no $2^{o(k)} \cdot n^{O(1)}$ algorithm for Vertex Cover on an n-vertex graph G.

Other problems

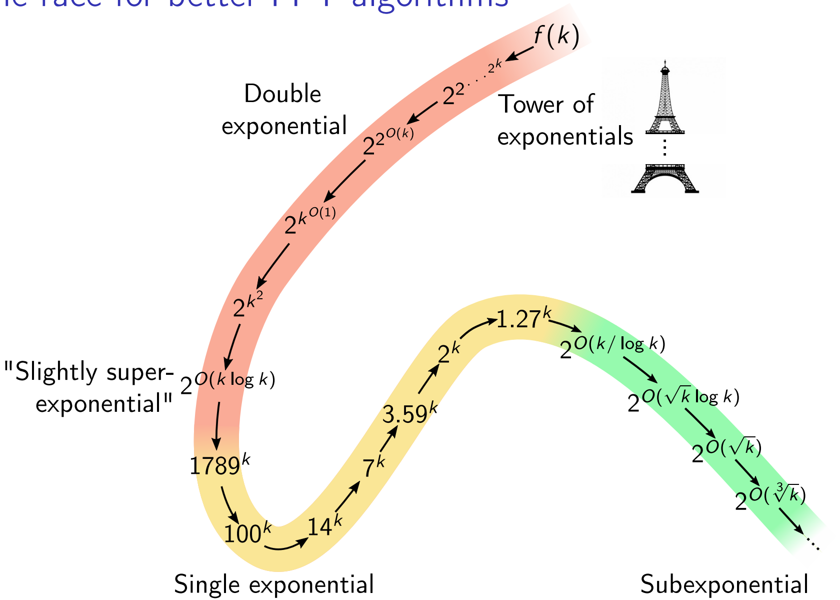
There are polytime reductions from 3SAT to many problems such that the reduction creates a graph with $O(n+m)$ vertices/edges.

Consequence: Assuming ETH, the following problems cannot be solved in time $2^{o(n)}$ and hence in time $2^{o(k)} \cdot n^{O(1)}$ (but $2^{O(k)} \cdot n^{O(1)}$ time algorithms are known):

- Vertex Cover
- Longest Cycle
- Feedback Vertex Set
- Multiway Cut
- Odd Cycle Transversal
- Steiner Tree
- ...

Seems to be the natural behavior of FPT problems?

The race for better FPT algorithms

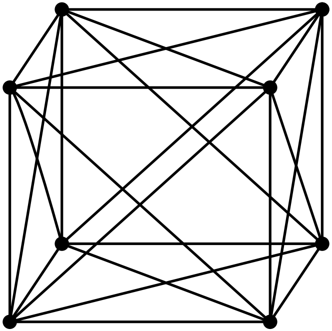


Edge Clique Cover

Edge Clique Cover: Given a graph G and an integer k, cover the edges of G with at most k cliques.

(the cliques need not be edge disjoint)

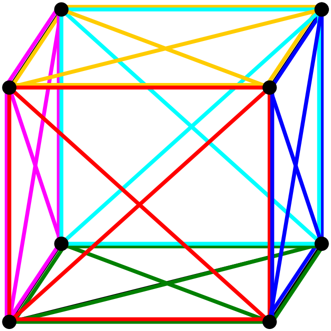
Equivalently: can G be represented as an intersection graph over a k element universe?



Edge Clique Cover

> Edge Clique Cover: Given a graph G and an integer k, cover the edges of G with at most k cliques.
>
> (the cliques need not be edge disjoint)

Equivalently: can G be represented as an intersection graph over a k element universe?



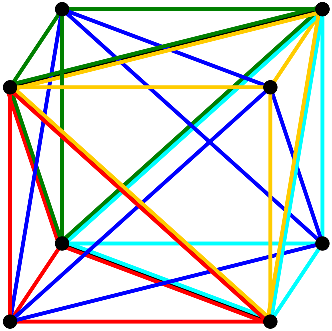
6 cliques

Edge Clique Cover

Edge Clique Cover: Given a graph G and an integer k, cover the edges of G with at most k cliques.

(the cliques need not be edge disjoint)

Equivalently: can G be represented as an intersection graph over a k element universe?



5 cliques

EDGE CLIQUE COVER

EDGE CLIQUE COVER: Given a graph G and an integer k, cover the edges of G with at most k cliques.

(the cliques need not be edge disjoint)

Simple algorithm (sketch)

- If two adjacent vertices have the same neighborhood ("twins"), then remove one of them.
- If there are no twins and isolated vertices, then $|V(G)| > 2^k$ implies that there is no solution.
- Use brute force.

Running time: $2^{2^{O(k)}} \cdot n^{O(1)}$ — double exponential dependence on k!

Edge Clique Cover

Edge Clique Cover: Given a graph G and an integer k, cover the edges of G with at most k cliques.

(the cliques need not be edge disjoint)

Double-exponential dependence on k cannot be avoided!

Theorem [Cygan, Pilipczuk, Pilipczuk 2013]

Assuming ETH, there is no $2^{2^{o(k)}} \cdot n^{O(1)}$ time algorithm for Edge Clique Cover.

Proof: Reduce an n-variable 3SAT instance into an instance of Edge Clique Cover with $k = O(\log n)$.

Slightly superexponential algorithms

Running time of the form $2^{O(k \log k)} \cdot n^{O(1)}$ appear naturally in parameterized algorithms usually because of one of two reasons:

1. Branching into k directions at most k times explores a search tree of size $k^k = 2^{O(k \log k)}$.
2. Trying $k! = 2^{O(k \log k)}$ permutations of k elements (or partitions, matchings, ...)

Can we avoid these steps and obtain $2^{O(k)} \cdot n^{O(1)}$ time algorithms?

Slightly superexponential algorithms

The improvement to $2^{O(k)}$ often required significant new ideas:

k-PATH:

$2^{O(k \log k)} \cdot n^{O(1)}$ using **representative sets** [Monien 1985]

$\Downarrow$

$2^{O(k)} \cdot n^{O(1)}$ using **color coding** [Alon, Yuster, Zwick 1995]

FEEDBACK VERTEX SET:

$2^{O(k \log k)} \cdot n^{O(1)}$ using **k-way branching** [Downey and Fellows 1995]

$\Downarrow$

$2^{O(k)} \cdot n^{O(1)}$ using **iterative compression** [Guo et al. 2005]

PLANAR SUBGRAPH ISOMORPHISM:

$2^{O(k \log k)} \cdot n^{O(1)}$ using **tree decompositions** [Eppstein et al. 1995]

$\Downarrow$

$2^{O(k)} \cdot n^{O(1)}$ using **sphere cut decompositions** [Dorn 2010]

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 $1 \leq i \leq k$.

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 $d(s, s_i) \leq d$ for every $1 \leq i \leq k$.

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 $d(s, s_i) \leq d$ for every $1 \leq i \leq k$.

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

Theorem [Gramm, Niedermeier, Rossmanith 2003]

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

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 $1 \leq i \leq k$.

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

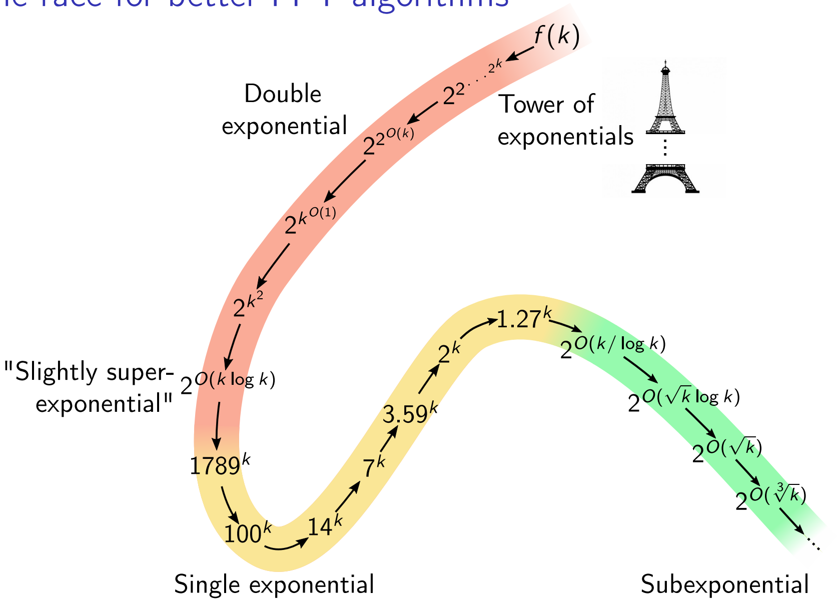
Theorem [Gramm, Niedermeier, Rossmanith 2003]

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

Theorem [Lokshtanov, M., Saurabh 2011]

Assuming ETH, Closest String has no $2^{o(d \log d)} n^{O(1)}$ algorithm.

The race for better FPT algorithms



Subexponential parameterized algorithms

There are two main domains where subexponential parameterized algorithms appear:

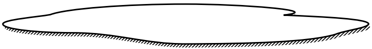
1. Some graph modification problems:
   - Chordal Completion [Fomin and Villanger 2013]
   - Interval Completion [Bliznets et al. 2016]
   - Unit Interval Completion [Bliznets et al. 2015]
   - Feedback Arc Set in Tournaments [Alon et al. 2009]

Subexponential parameterized algorithms

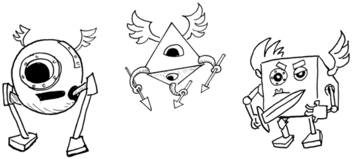
There are two main domains where subexponential parameterized algorithms appear:

1. Some graph modification problems:
   - Chordal Completion [Fomin and Villanger 2013]
   - Interval Completion [Bliznets et al. 2016]
   - Unit Interval Completion [Bliznets et al. 2015]
   - Feedback Arc Set in Tournaments [Alon et al. 2009]
2. "Square root phenomenon" for planar graphs and geometric objects: most NP-hard problems are easier and usually exactly by a square root factor.

Planar graphs



Geometric objects



Square root phenomenon for planar graphs

NP-hard problems become easier on planar graphs and usually exactly by a square root factor.

The running time is still exponential, but significantly smaller:

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

3-Coloring, Independent Set, Vertex Cover, Dominating Set, Hamiltonian Cycle, k-Path, ...

Other planar subexponential algorithms

Many other result were obtained using problem-specific techniques:

- Subgraph Isomorphism
  for connected bounded-degree patterns [Fomin et al. 2016]
- Subset TSP [Klein and M. 2014]
- Directed Subset TSP [M., Pilipczuk, Pilipczuk 2018]
- Bipartite Deletion [Lokshtanov, Saurabh, Wahlström 2012]

Other planar subexponential algorithms

Many other result were obtained using problem-specific techniques:

- Subgraph Isomorphism
  for connected bounded-degree patterns [Fomin et al. 2016]
- Subset TSP [Klein and M. 2014]
- Directed Subset TSP [M., Pilipczuk, Pilipczuk 2018]
- Bipartite Deletion [Lokshtanov, Saurabh, Wahlström 2012]

A recent negative result:

Steiner Tree with k terminals

- can be solved in time $2^{O(k)} \cdot n^{O(1)}$ in **general** graphs,
  [Dreyfus and Wagner 1971]
- cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$ in **planar undirected** graphs (assuming the ETH).
  [M., Pilipczuk, Pilipczuk 2018]

Shift of focus

qualitative question

FPT or W[1]-hard?

FPT

W[1]-hard

quantitative question

What is the best possible multiplier $f(k)$ in the running time $f(k) \cdot n^{O(1)}$?

2^k? 1.0001^k? $2^{\sqrt{k}}$?

What is the best possible exponent $g(k)$ in the running time $f(k) \cdot n^{g(k)}$?

$n^{O(k)}$? $n^{\log k}$? $n^{\log \log k}$?

Better algorithms for W[1]-hard problems

- $O(n^k)$ algorithm for k-Clique by brute force.
- $O(n^{0.79k})$ algorithms using fast matrix multiplication.
- W[1]-hardness of k-Clique gives evidence that there is no $f(k) \cdot n^{O(1)}$ time algorithm.
- But what about improvements of the exponent $O(k)$?

$n^{\sqrt{k}}$

$n^{k/\log\log k}$

$n^{\log k}$

$n^{\sqrt{k}}$

$2^{2^k} \cdot n^{\log\log\log k}$

Better algorithms for W[1]-hard problems

- $O(n^k)$ algorithm for k-Clique by brute force.
- $O(n^{0.79k})$ algorithms using fast matrix multiplication.
- W[1]-hardness of k-Clique gives evidence that there is no $f(k) \cdot n^{O(1)}$ time algorithm.
- But what about improvements of the exponent $O(k)$?

Theorem [Chen et al. 2004]

Assuming ETH, k-Clique has no $f(k) \cdot n^{o(k)}$ time algorithm for any computable function f.

Better algorithms for W[1]-hard problems

- $O(n^k)$ algorithm for DOMINATING SET by brute force.
- W[1]-hardness of DOMINATING SET gives evidence that there is no $f(k) \cdot n^{O(1)}$ time algorithm.
- But what about improvements of the exponent $O(k)$?

$n^{\sqrt{k}}$ $n^{k/\log\log k}$ $n^{0.01k}$ $2^{2^k} \cdot n^{0.99k}$ $n^{\log\log\log k}$

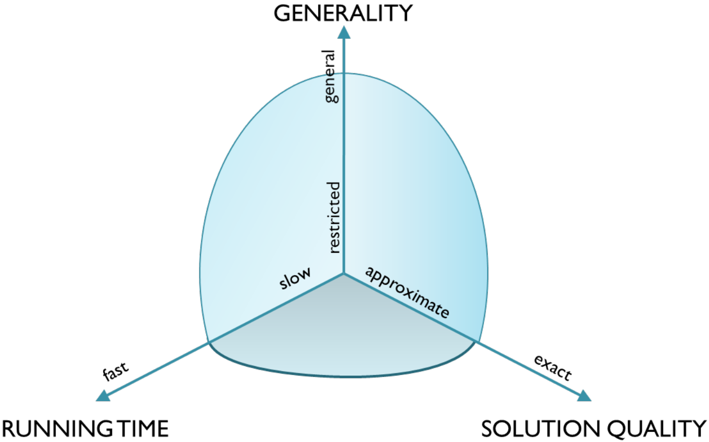
Better algorithms for W[1]-hard problems

- $O(n^k)$ algorithm for Dominating Set by brute force.
- W[1]-hardness of Dominating Set gives evidence that there is no $f(k) \cdot n^{O(1)}$ time algorithm.
- But what about improvements of the exponent $O(k)$?

Theorem [Pătraşcu and Williams 2010]

Assuming SETH, Dominating Set has no $f(k) \cdot n^{k-\epsilon}$ time algorithm for any $\epsilon > 0$ and computable function f.

Dimensions


GENERALITY
general
restricted
slow
approximate
fast
exact
RUNNING TIME
SOLUTION QUALITY

From general to special

A major theme in the theoretical literature: consider restricted versions of hard problems.

- Restriction to graph classes of practical or theoretical interest.
- Restricting the number of special objects.
- Restricted type of constraints.
- ...

More restricted problem	$\Rightarrow$	More possibility for algorithmic ideas

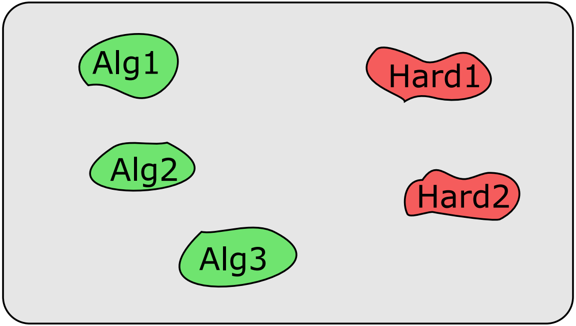
From general to special

A major theme in the theoretical literature: consider restricted versions of hard problems.

- Restriction to graph classes of practical or theoretical interest.
- Restricting the number of special objects.
- Restricted type of constraints.
- ...

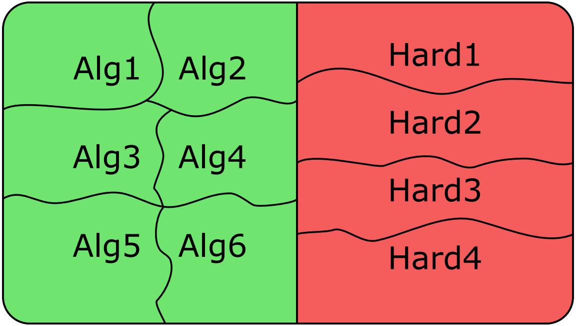
> Find every relevant algorithmic idea by exploring every possible tractable restriction.

Mapping the complexity landscape



From partial results...

Mapping the complexity landscape



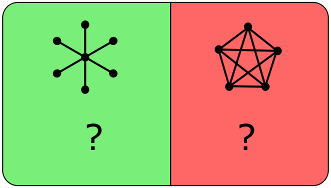
...to a complete **dichotomy.**

Goal:
A complete classification explaining the complexity of every restricted problem by a few algorithms and hardness results.

Finding patterns

Basic problem: find/count/pack/cover occurrences of a specific fixed pattern in a graph.
[graph transformations, chemical structures, pattern recognition, protein-protein interactions...]

Some patterns are easy to handle...



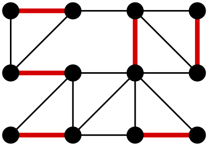
Some patterns are hard to handle...

Goal:
Classify the complexity for all types of patterns and discover all the relevant algorithmic techniques.

Factor problems

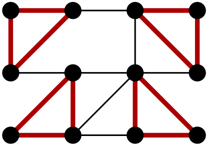
PERFECT MATCHING
Input: n-vertex graph G.
Task: find $n/2$ vertex-disjoint edges.

Polynomial-time solvable [Edmonds 1961].



TRIANGLE FACTOR
Input: n-vertex graph G.
Task: find $n/3$ vertex-disjoint triangles.

NP-complete [Karp 1975]



Factor problems

H-FACTOR
Input: n-vertex graph G.
Task: find $n/|V(H)|$ vertex-disjoint copies of H in G.

Polynomial-time solvable for $H = K_2$ and NP-hard for $H = K_3$.

Which graphs H make H-FACTOR easy and which graphs make it hard?

Factor problems

H-FACTOR
Input: n-vertex graph G.
Task: find $n/|V(H)|$ vertex-disjoint copies of H in G.

Polynomial-time solvable for $H = K_2$ and NP-hard for $H = K_3$.

Which graphs H make H-FACTOR easy and which graphs make it hard?

Theorem [Kirkpatrick and Hell 1978]

H-FACTOR is NP-hard for every connected graph H with at least 3 vertices.

Factor problems

Instead of publishing

Kirkpatrick and Hell: NP-completeness of packing cycles. 1978.
Kirkpatrick and Hell: NP-completeness of packing trees. 1979.
Kirkpatrick and Hell: NP-completeness of packing stars. 1980.
Kirkpatrick and Hell: NP-completeness of packing wheels. 1981.
Kirkpatrick and Hell: NP-completeness of packing Petersen graphs. 1982.
Kirkpatrick and Hell: NP-completeness of packing Starfish graphs. 1983.
Kirkpatrick and Hell: NP-completeness of packing Jaws. 1984.

$\vdots$

they only published

Kirkpatrick and Hell: On the Completeness of a Generalized Matching Problem. 1978

Counting patterns

#*H*-SUBGRAPH
Input: n-vertex graph G.
Task: count the number of copies of H in G as subgraph.

Which pattern graphs H make this problem polynomial-time solvable?

Counting patterns

#H-SUBGRAPH
Input: n-vertex graph G.
Task: count the number of copies of H in G as subgraph.

Which pattern graphs H make this problem polynomial-time solvable?

Trivial answer: Polynomial-time solvable for every fixed H with k vertices in $n^{O(k)}$ time.

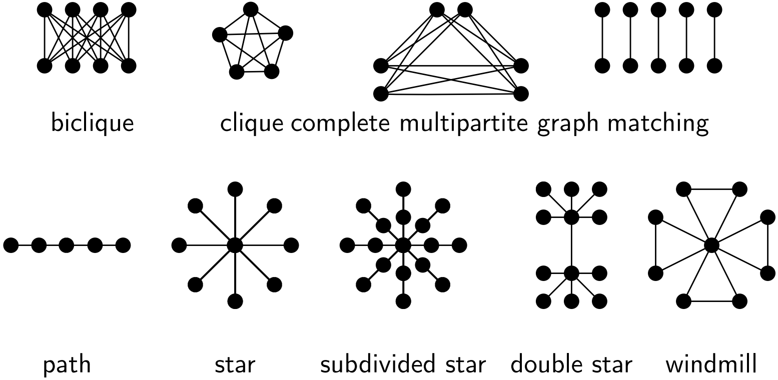
Better questions:

- What *classes* of patterns are easy?
- What is the exact exponent of n for a given H?

Counting patterns

Main question

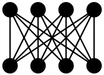
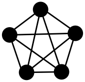
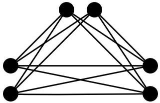
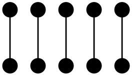
Which type of subgraph patterns are easy to count?



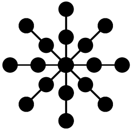
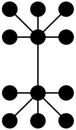
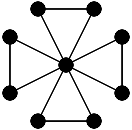
Counting patterns

Main question

Which type of subgraph patterns are easy to count?

biclique clique complete multipartite graph matching

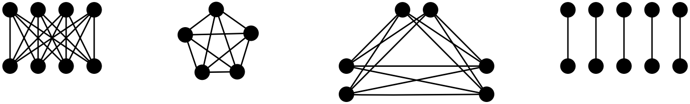
    

path star subdivided star double star windmill

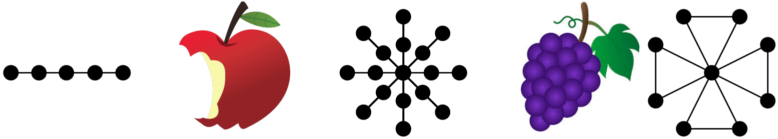
Counting patterns

Main question

Which type of subgraph patterns are easy to count?



biclique clique complete multipartite graph matching

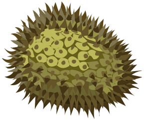
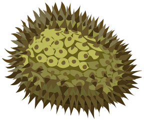
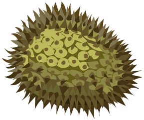
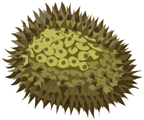


path star subdivided star double star windmill

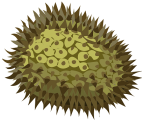
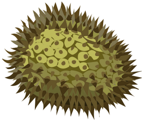
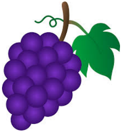
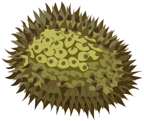
Counting patterns

Main question

Which type of subgraph patterns are easy to count?

biclique clique complete multipartite graph matching

path star subdivided star double star windmill

Counting subgraphs

Vertex cover number of H determines the complexity of counting copies of H:

- $n^{\mathrm{vc}(H)+O(1)}$ upper bound.
  [Multiple references]
- $\Omega(n^{\gamma \cdot \mathrm{vc}(H)/\log \mathrm{vc}(H)})$ lower bound.
  [Curticapean, Dell, M. 2017]

If we restrict the problem to a class $\mathcal{H}$ of patterns:

- If $\mathcal{H}$ has bounded vertex cover number (e.g, stars, double stars, ...), then the problem is polynomial-time solvable.
- If $\mathcal{H}$ has unbounded vertex cover number (e.g, cliques, paths, matchings, disjoint triangles, ...), then the problem is **not** polynomial-time solvable (assuming ETH).

Summary

- There are more precise questions than just polynomial time vs. NP-hardness...
- ...and in many cases, we have precise answers.
- Running time, generality, solution quality.
- Algorithm design and computational complexity have healthy influence on each other.

Summary

- There are more precise questions than just polynomial time vs. NP-hardness...
- ...and in many cases, we have precise answers.
- Running time, generality, solution quality.
- Algorithm design and computational complexity have healthy influence on each other.

Think of lower bounds
when designing algorithms

Summary

- There are more precise questions than just polynomial time vs. NP-hardness...
- ...and in many cases, we have precise answers.
- Running time, generality, solution quality.
- Algorithm design and computational complexity have healthy influence on each other.

Think of lower bounds
when designing algorithms

Think of algorithms
when doing lower bounds