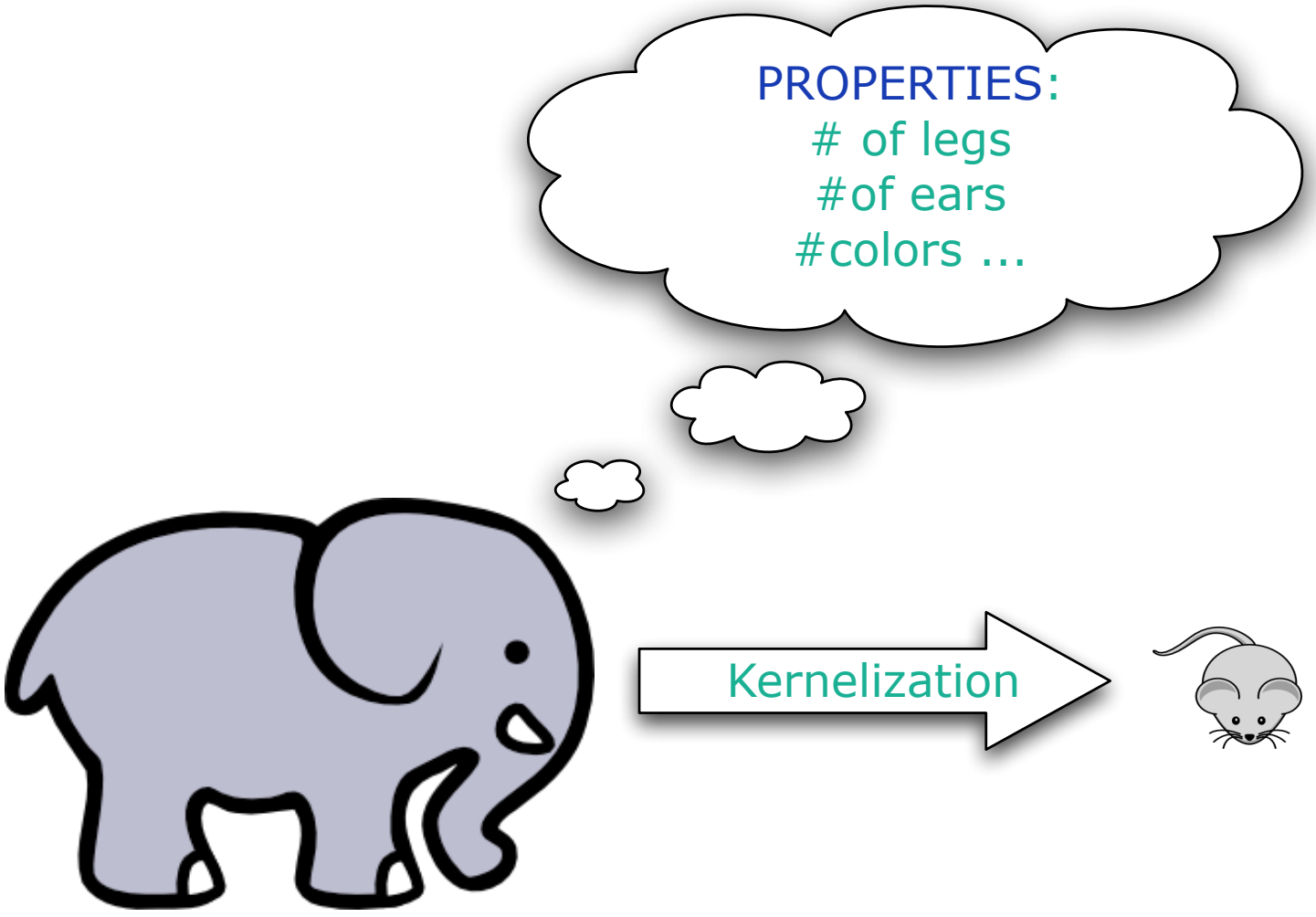
FEDOR V. FOMIN

Introduction to Kernelization

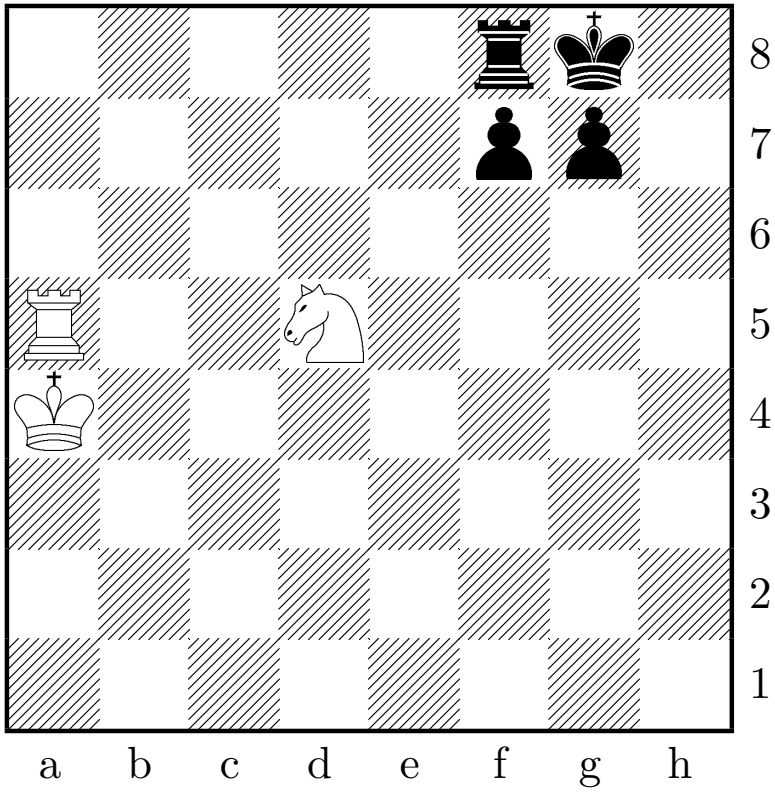
Sophia Antipolis
Colloquium Jacques Morgenstern

17.11.2011

Outline of the talk

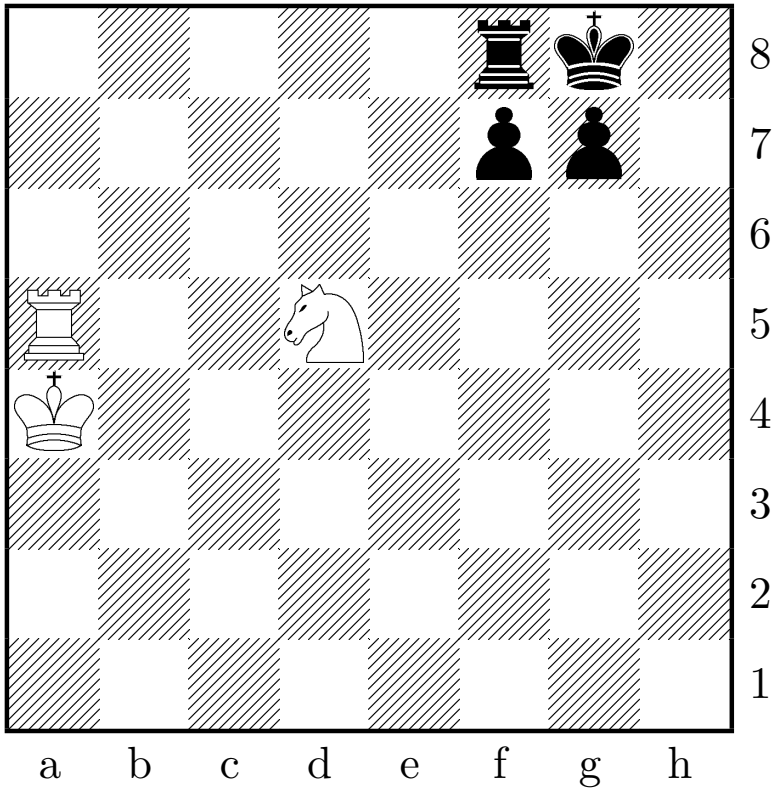


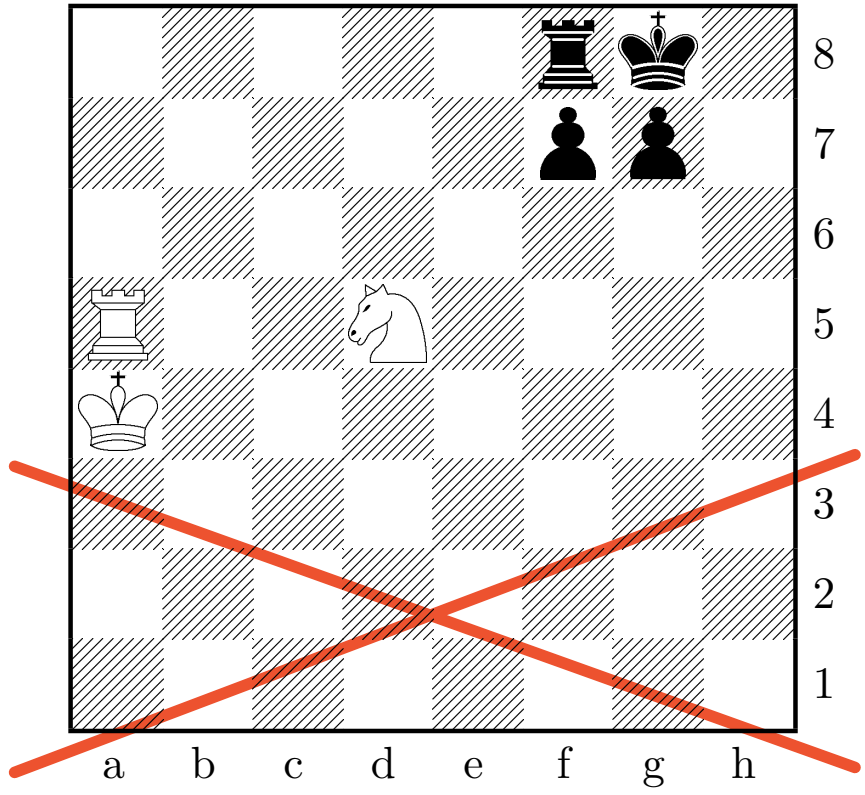
Preprocessing. Chess Example



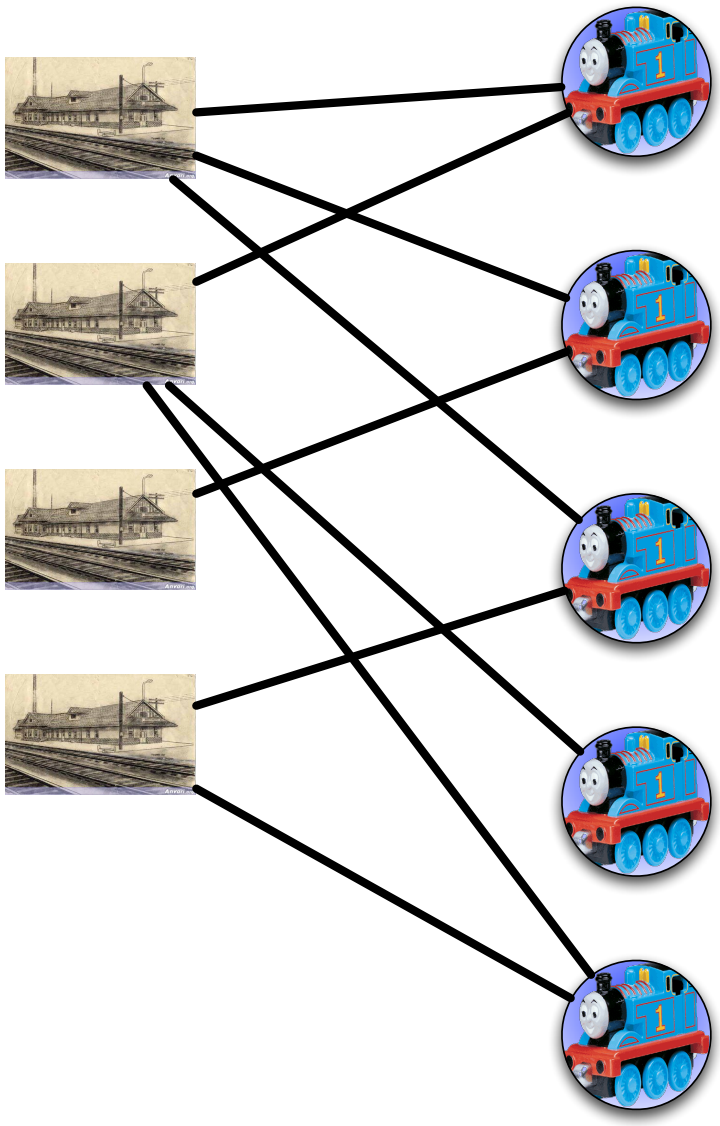
– White to start and to mate in two moves

Preprocessing. Chess Example



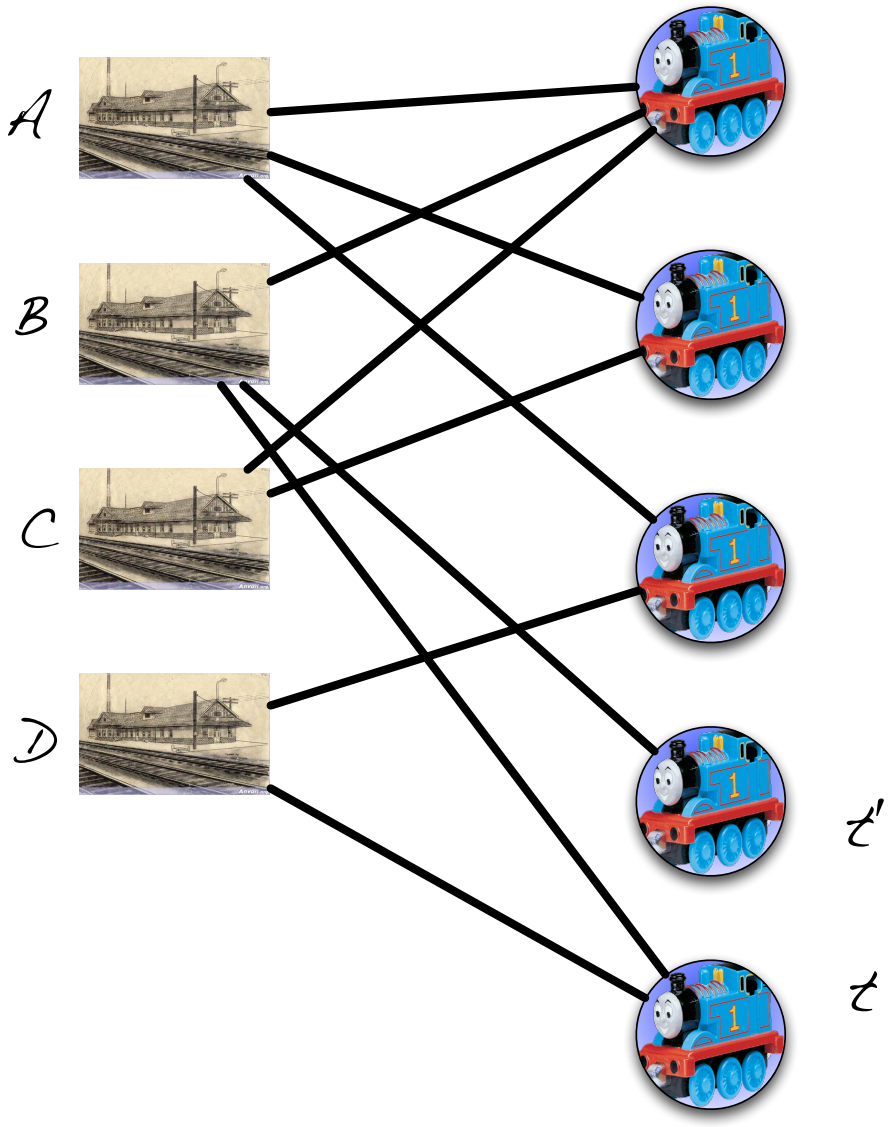


Preprocessing. Train management



– We want a minimum-sized subset of stations S such that every train stops at least at one station from S

Preprocessing. Train management



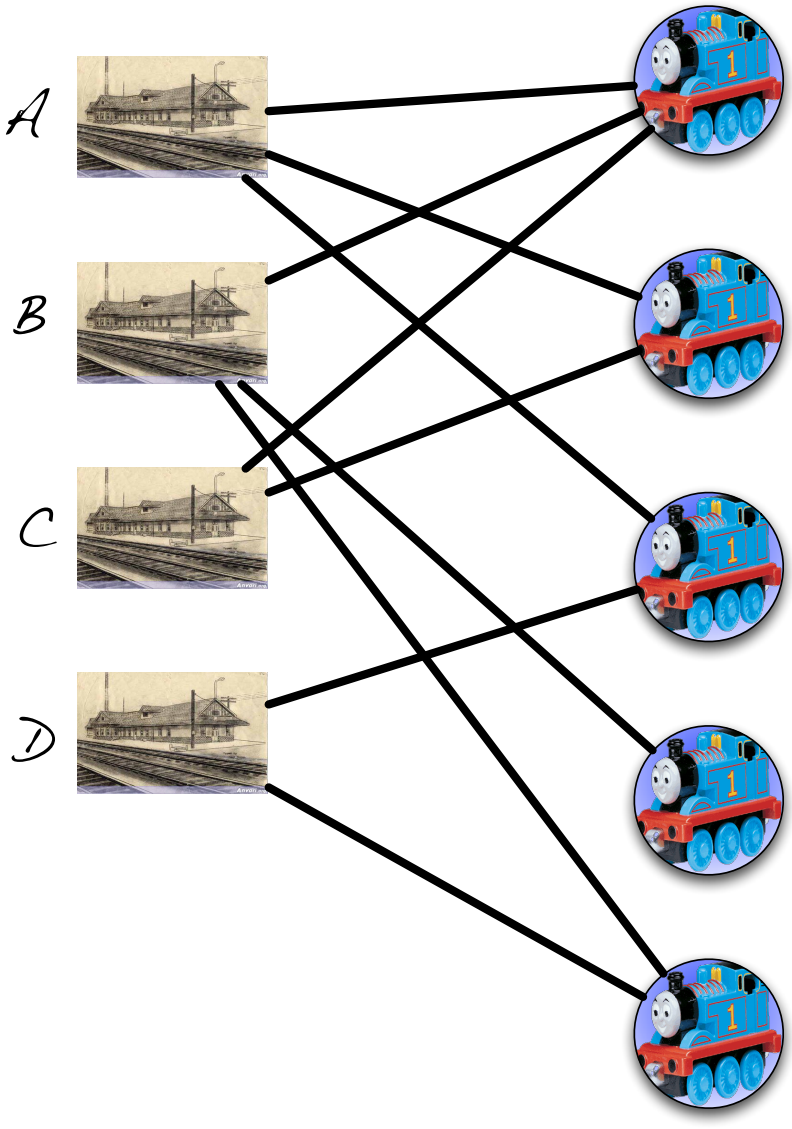
Reduction Rules:

- Rule 1 If $S(t) \subseteq S(t')$, then remove t

$S(t)=\{B,D\}$

$S(t')=\{B\}$

Preprocessing. Train management



Reduction Rules:

- Rule 2 If $T(s) \subseteq T(s')$,

then remove s

Preprocessing. Train management

K.Weihe, ALEX, 1998: Similar preprocessing for real-world data from the German and European train schedules (25.000 stations, 154.000 trains and 160.000 single train stops) the data reduction merely took a few minutes to reduce the original, huge input graph into a graph consisting of disjoint components of size at most 50.

Preprocessing is ubiquitous

- Commercial linear program solvers like CPLEX
- Navigation systems
- Microarray data analysis for the classification of cancer types
- ...

Analysis of Algorithms

- Powerful tools developed since 1960s
- Theory of NP-completeness

Analysis of Algorithms. Naive question

- Take your favourite NP-complete problem
- Is there preprocessing algorithm that guarantee to reduce every instance of your problem, say by 5%?

Analysis of Algorithms. Naive question

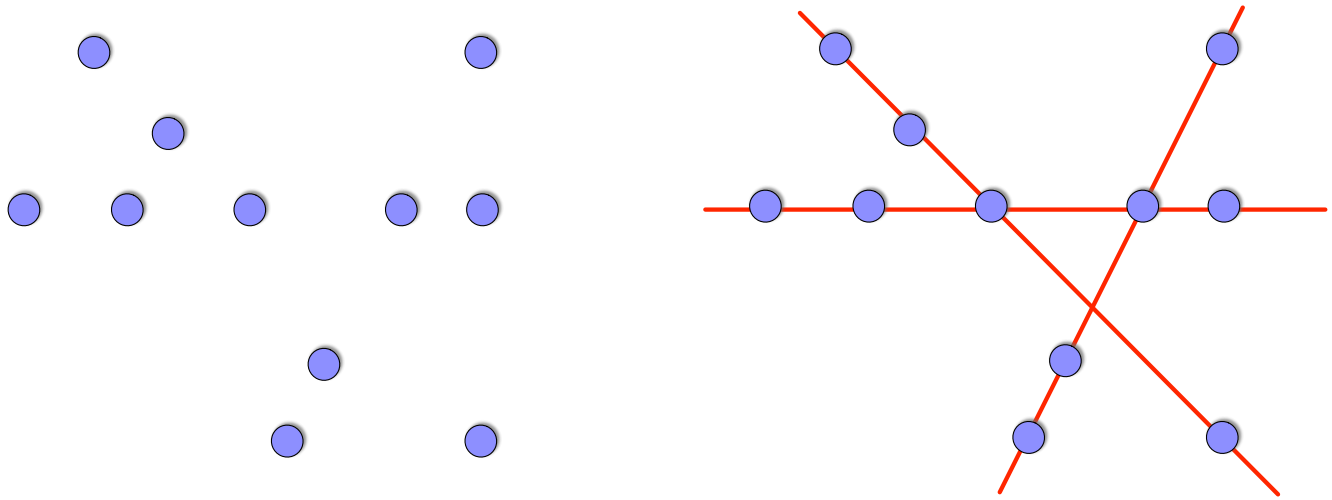
- Take your favourite NP-complete problem
- Is there preprocessing algorithm that guarantee to reduce every instance of your problem, say by 5%?
- Well, that would be very strange!!!

Analysis of Algorithms. Naive question

- Take your favourite NP-complete problem
- Is there preprocessing algorithm that guarantee to reduce every instance of your problem, say by 5%?
- Well, that would be very strange!!!
- At first glance, no interesting theory for preprocessing of hard problems!

Covering Points with Lines

Task: Given a set P of n points in the plane and an integer k, find k lines that cover all the points.



Note: We can assume that every line of the solution covers at least 2 points, thus there are at most n^2 candidate lines.

Covering Points with Lines

Reduction Rule If there is a line L covering more than k points, remove all points covered by L and reduce the parameter k by one.

Covering Points with Lines

Reduction Rule If there is a line L covering more than k points, remove all points covered by L and reduce the parameter k by one.

Why this rule is sound?

Covering Points with Lines

At every step of Reduction Rule we obtain a problem with a smaller number of points. Thus we

- either end up with the problem with no points left, and in this case we solved the problem; YES!

Covering Points with Lines

At every step of Reduction Rule we obtain a problem with a smaller number of points. Thus we

- either end up with the problem with no points left, and in this case we solved the problem; YES!
- or the parameter k is zero but some points are left, in this case the problem does not have solution; NO!

Covering Points with Lines

At every step of Reduction Rule we obtain a problem with a smaller number of points. Thus we

- either end up with the problem with no points left, and in this case we solved the problem; YES!
- or the parameter k is zero but some points are left, in this case the problem does not have solution; NO!
- or we arrive at the problem for which our Reduction Rule cannot be applied. What happens here?

Covering Points with Lines

Reduction Rule If there is a line L covering more than k points, remove all points covered by L and reduce the parameter k by one.

If Rule cannot be applied, WE HAVE AT MOST k^2 POINTS!

So we have an algorithm that

Input: An instance of our (NP-complete) problem of size n, and an integer k.
Output: Either correct solution, or an equivalent instance.
Properties of the algorithm and the reduced instance

- It runs in time $O(n^2)$ [polynomial time]
- Outputs an equivalent instance of size k^2 [the size of reduced instance depends only on k].

This is what kernels are about

Input: An instance of our (NP-complete) problem of size n, and an integer k.
Output: Either correct solution, or an equivalent instance.
Properties of the algorithm and the reduced instance

- It runs in polynomial time $O(n^c)$, c some constant
- Outputs an equivalent instance of size $f(k)$, $f(k)$ is some function.

Classical complexity

A brief review:

- We usually aim for **polynomial-time** algorithms: the running time is $O(n^c)$, where n is the input size.
- Classical polynomial-time algorithms: shortest path, mathching, minimum spanning tree, 2SAT, convext hull, planar drawing, linear programming, etc.
- It is unlikely that polynomial-time algorithms exist for **NP-hard** problems.

Classical complexity

- Unfortunately, many problems of interest are NP-hard: Hamiltonian cycle, 3-coloring, 3SAT, etc.
- We expect that these problems can be solved only in exponential time (i.e., c^n).

Can we say anything nontrivial about NP-hard problems?

Parameterized complexity

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.

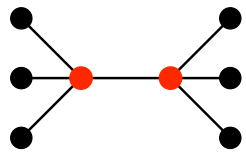
Parameterized complexity

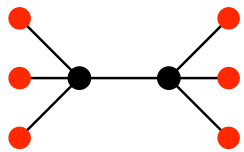
What can be the parameter k?

- The size k of the solution we are looking for.
- The maximum degree of the input graph.
- The diameter of the input graph.
- The length of clauses in the input Boolean formula.
- ...

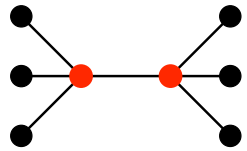
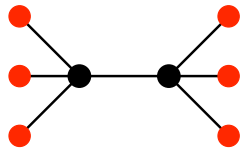
Parameterized complexity

	Min Vertex Cover	Max Independent Set
Problem:		
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?

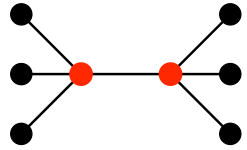
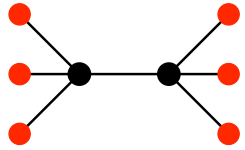




Parameterized complexity

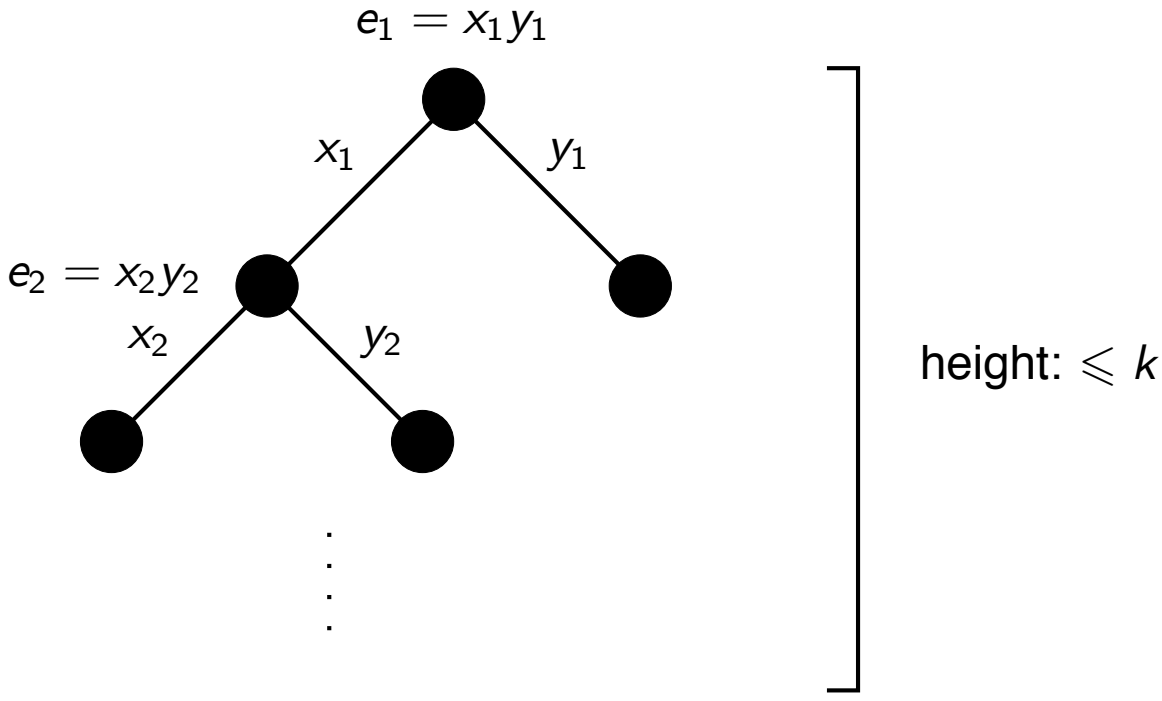
Problem:	Min Vertex Cover	Max Independent Set
Input:	Graph G, integer k	Graph G, integer k
Question:	Is it possible to cover the edges with k vertices?	Is it possible to find k independent vertices?
		
Complexity:	NP-complete	NP-complete

Parameterized complexity

Problem:	Min Vertex Cover	Max 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
Complete Enumeration:	$O(n^k)$ possibilities $O(2^k n^2)$ algorithm exists 	$O(n^k)$ possibilities No $n^{o(k)}$ algorithm known 

Bounded search tree method

Algorithm for Minimum Vertex Cover:



Height of the search tree is $\leq k \Rightarrow$ number of leaves is $\leq 2^k \Rightarrow$ complete search requires $2^k \cdot$ poly steps.

Fixed-parameter tractability

Definition: A *parameterization* of a decision problem is a function that assigns an integer parameter k to each input instance x.

The parameter can be

- explicit in the input (for example, if the parameter is the integer k appearing in the input (G, k) of Vertex Cover), or
- implicit in the input (for example, if the parameter is the diameter d of the input graph G).

Fixed-parameter tractability

Definition: A *parameterization* of a decision problem is a function that assigns an integer parameter k to each input instance x.

The parameter can be

- explicit in the input (for example, if the parameter is the integer k appearing in the input (G, k) of Vertex Cover), or
- implicit in the input (for example, if the parameter is the diameter d of the input graph G).

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

Example: Minimum Vertex Cover parameterized by the required size k is FPT: we have seen that it can be solved in time $O(2^k + n^2)$.

Better algorithms are known: e.g, $O(1.2832^k k + k|V|)$.

Main goal of parameterized complexity: to find FPT problems.

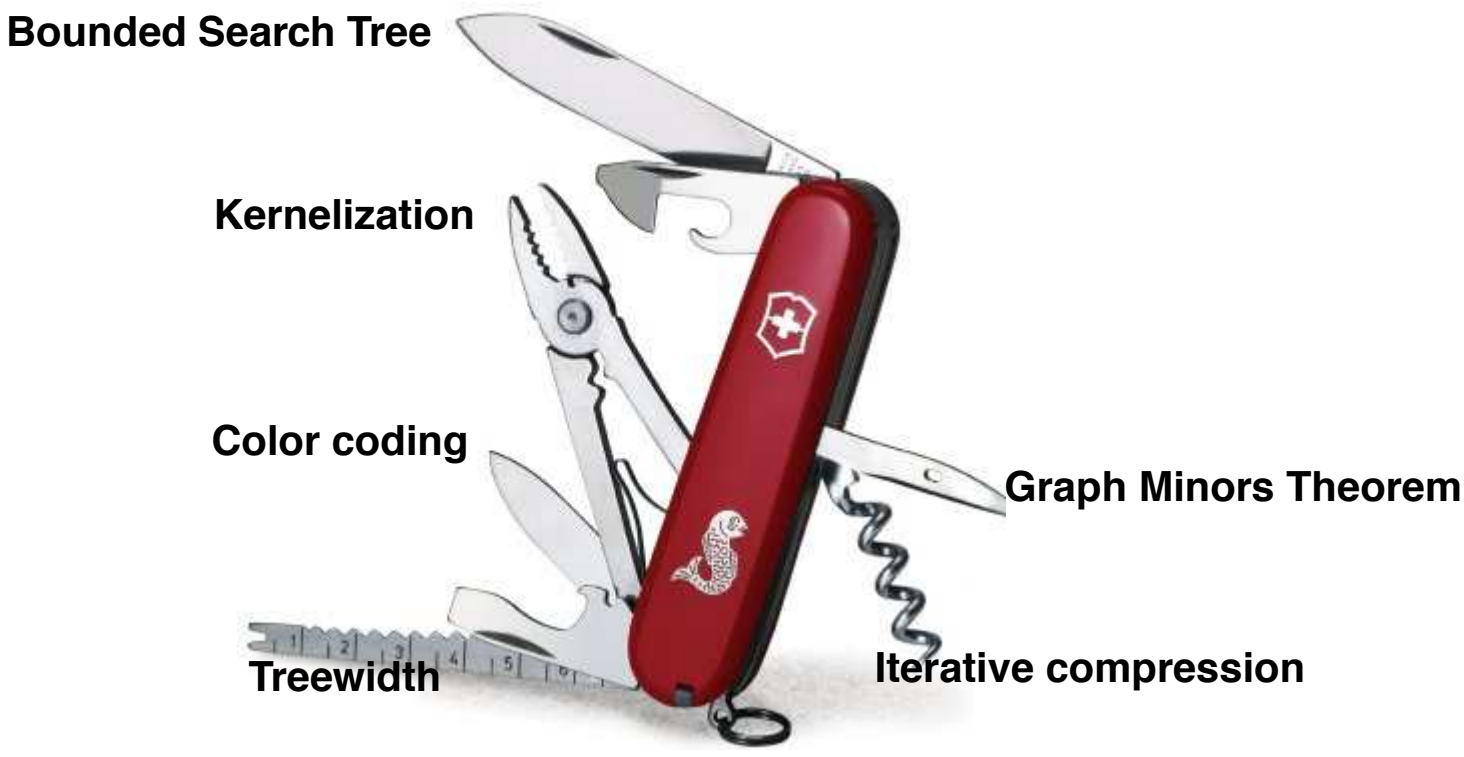
FPT problems

Examples of NP-hard problems that are FPT:

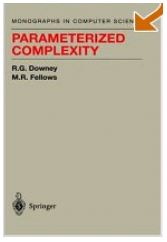
- Finding a vertex cover of size k.
- Finding a path of length k.
- Finding k disjoint triangles.
- Drawing the graph in the plane with k edge crossings.
- Finding disjoint paths that connect k pairs of points.
- ...

FPT algorithmic techniques

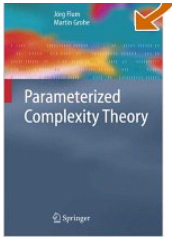
- Significant advances in the past 20 years or so (especially in recent years).
- Powerful toolbox for designing FPT algorithms:



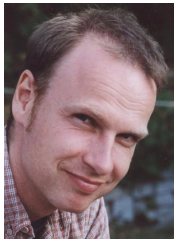
Books

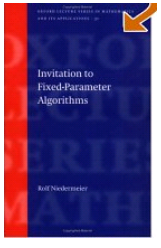


Downey-Fellows: Parameterized Complexity, Springer, 1999



Flum-Grohe: Parameterized Complexity Theory, Springer, 2006





Niedermeier: Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.

Kernelization

Definition: *Kernelization* is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

Kernelization

Definition: *Kernelization* is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

(I', k') is *kernel*.

Kernelization

Definition: *Kernelization* is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

(I', k') is *kernel*.

Our example with n points and lines.

- Instance (I, k) with n points and parameter k
- New equivalent instance (I', k') with $k' \leq k$ and I' consisting of at most k^2 points: a kernel with k^2 points.

TWO BASIC QUESIONS:

--- Does the problem have a kernel?

--- How large the kernel can be?

When the problem has a kernel?

Definition: *Kernelization* is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

When the problem has a kernel?

Definition: *Kernelization* is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

Simple fact: If a problem has a kernelization algorithm, then it is FPT.

Proof: Solve the instance (I', k') by brute force.

When the problem has a kernel?

Definition: *Kernelization* is a polynomial-time transformation that maps an instance (I, k) to an instance (I', k') such that

- (I, k) is a yes-instance if and only if (I', k') is a yes-instance,
- $k' \leq k$, and
- $|I'| \leq f(k)$ for some function $f(k)$.

Simple fact: If a problem has a kernelization algorithm, then it is FPT.
Proof: Solve the instance (I', k') by brute force.

Converse: Every FPT problem has a kernelization algorithm.
Proof: Suppose there is an $f(k)n^c$ algorithm for the problem.

- If $f(k) \leq n$, then solve the instance in time $f(k)n^c \leq n^{c+1}$, and output a trivial yes- or no-instance.
- If $n < f(k)$, then we are done: a kernel of size $f(k)$ is obtained.

TWO BASIC QUESIONS:

--- Does the problem have a kernel?

Problem has a kernel <=> Problem is in FPT

--- How large the kernel can be?

Polynomial kernels via exponential kernels

What's next

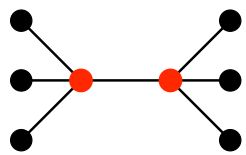
Two kernels for Vertex Cover

I: Naive approach

II: Crown Decomposition

Kernelization for Vertex Cover

Problem: Min Vertex Cover
Input: Graph G, integer k
Question: Is it possible to cover the edges with k vertices?



Kernelization for Vertex Cover

General strategy: We devise a list of reduction rules, and show that if none of the rules can be applied and the size of the instance is still larger than $f(k)$, then the answer is trivial.

Reduction rules for Vertex Cover instance (G, k):

Rule 1: If v is an isolated vertex $\Rightarrow (G \setminus v, k)$
Rule 2: If $d(v) > k \Rightarrow (G \setminus v, k-1)$

Kernelization for Vertex Cover

General strategy: We devise a list of reduction rules, and show that if none of the rules can be applied and the size of the instance is still larger than $f(k)$, then the answer is trivial.

Reduction rules for Vertex Cover instance (G, k):

Rule 1: If v is an isolated vertex $\Rightarrow (G \setminus v, k)$
Rule 2: If $d(v) > k \Rightarrow (G \setminus v, k-1)$

If neither Rule 1 nor Rule 2 can be applied:

- If $|V(G)| > k(k+1) \Rightarrow$ There is no solution (every vertex should be the neighbor of at least one vertex of the cover).
- Otherwise, $|V(G)| \leq k(k+1)$ and we have a $k(k+1)$ vertex kernel.

Kernelization for Vertex Cover

Let us add a third rule:

Rule 1: If v is an isolated vertex $\Rightarrow (G \setminus v, k)$
Rule 2: If $d(v) > k \Rightarrow (G \setminus v, k-1)$
Rule 3: If $d(v) = 1$, then we can assume that its neighbor u is in the solution $\Rightarrow (G \setminus (u \cup v), k-1)$.

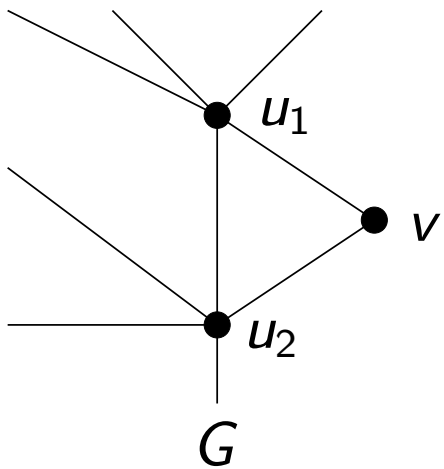
If none of the rules can be applied, then every vertex has degree at least 2.
$\Rightarrow |V(G)| \leq |E(G)|$

- If $|E(G)| > k^2 \Rightarrow$ There is no solution (each vertex of the solution can cover at most k edges).
- Otherwise, $|V(G)| \leq |E(G)| \leq k^2$ and we have a k^2 vertex kernel.

Kernelization for Vertex Cover

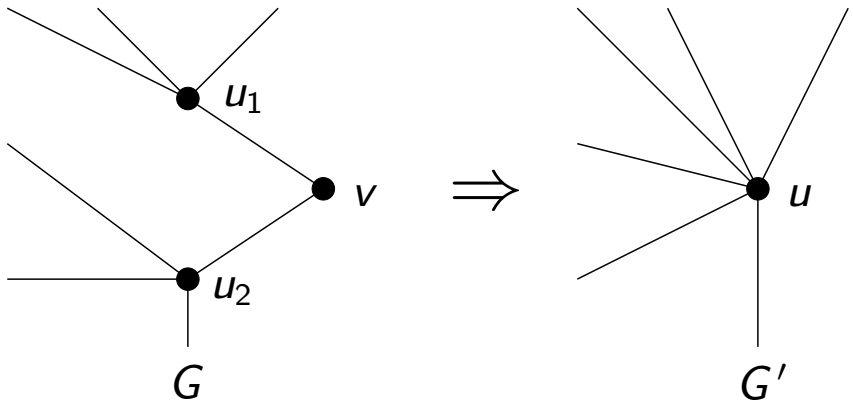
Let us add a fourth rule:

Rule 4a: If v has degree 2, and its neighbors u_1 and u_2 are adjacent, then we can assume that u_1, u_2 are in the solution $\Rightarrow$ $(G \setminus \{u_1, u_2, v\}, k-2)$.



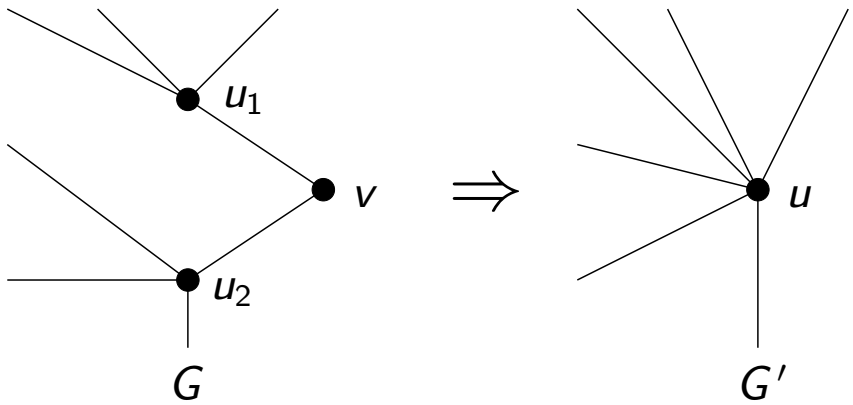
Kernelization for Vertex Cover

Rule 4b: If v has degree 2, then G' is obtained by identifying the two neighbors of v and deleting $v \Rightarrow (G', k-1)$.



Kernelization for Vertex Cover

Rule 4b: If v has degree 2, then G' is obtained by identifying the two neighbors of v and deleting $v \Rightarrow (G', k-1)$.

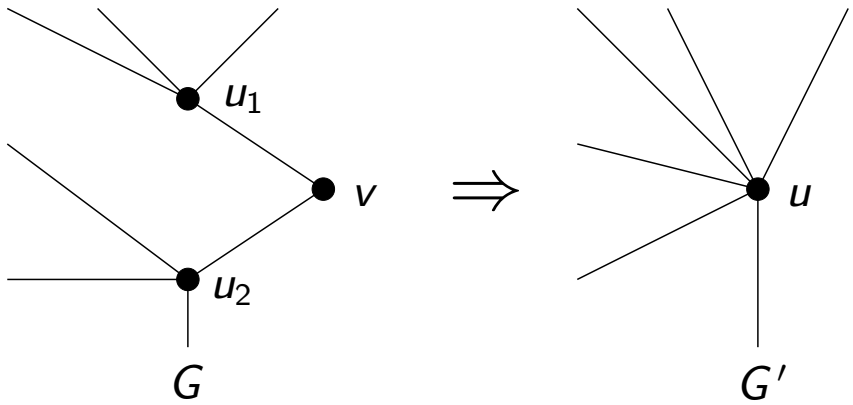


Correctness:

- Let S' be a vertex cover of size $k-1$ for G'.
- If $u \in S' \Rightarrow (S' \setminus u) \cup \{u_1, u_2\}$ is a vertex cover of size k for G.
- If $u \notin S' \Rightarrow S' \cup v$ is a vertex cover of size k for G.

Kernelization for Vertex Cover

Rule 4b: If v has degree 2, then G' is obtained by identifying the two neighbors of v and deleting $v \Rightarrow (G', k-1)$.

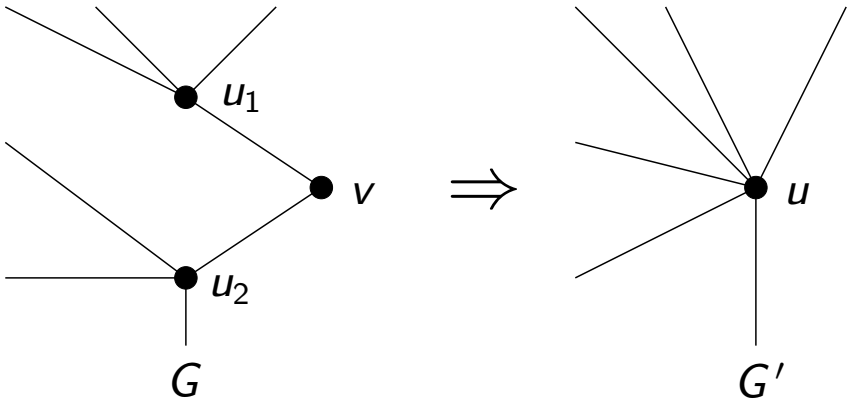


Correctness:

- Let S be a vertex cover of size k for G.
- If $u_1, u_2 \in S \Rightarrow (S \setminus \{u_1, u_2, v\}) \cup u$ is a vertex cover of size $k-1$ for G'.
- If exactly one of u_1 and u_2 is in S, then $v \in S \Rightarrow$ $(S \setminus \{u_1, u_2, v\}) \cup u$ is a vertex cover of size $k-1$ for G'.
- If $u_1, u_2 \notin S$, then $v \in S \Rightarrow (S \setminus v)$ is a vertex cover of size $k-1$ for G'.

Kernelization for Vertex Cover

Rule 4b: If v has degree 2, then G' is obtained by identifying the two neighbors of v and deleting $v \Rightarrow (G', k-1)$.



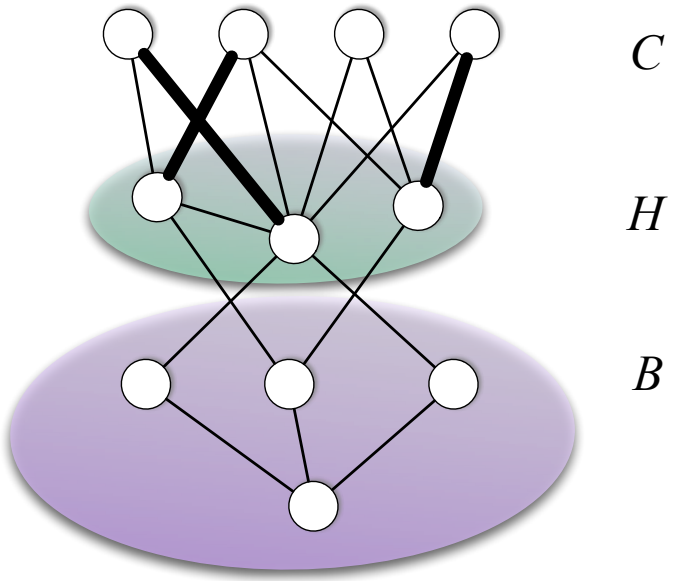
Kernel size:

- If $|E(G)| > k^2 \Rightarrow$ There is no solution (each vertex of the solution can cover at most k edges).
- Otherwise, $|V(G)| \leq 2|E(G)|/3 \leq \frac{2}{3}k^2$ and we have a $\frac{2}{3}k^2$ vertex kernel.

Crown Reduction

Definition: A **crown decomposition** is a partition $C \cup H \cup B$ of the vertices such that

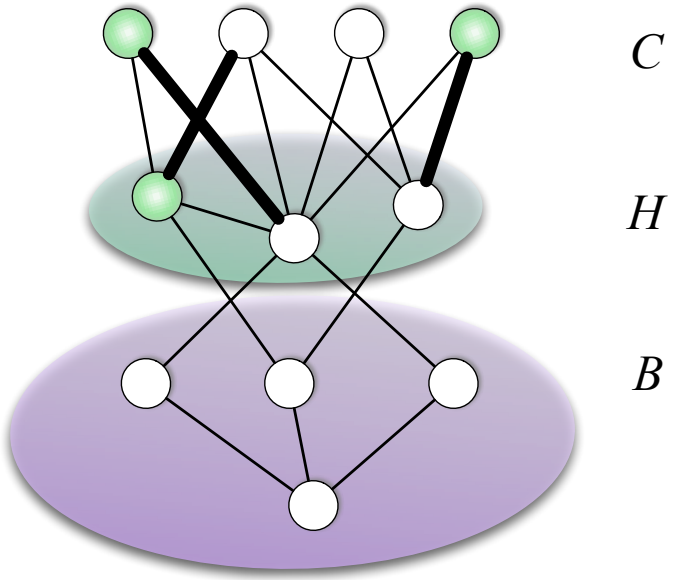
- C is an independent set,
- there is no edge between C and B,
- there is a matching between C and H that covers H.



Crown Reduction

Definition: A **crown decomposition** is a partition $C \cup H \cup B$ of the vertices such that

- C is an independent set,
- there is no edge between C and B,
- there is a matching between C and H that covers H.

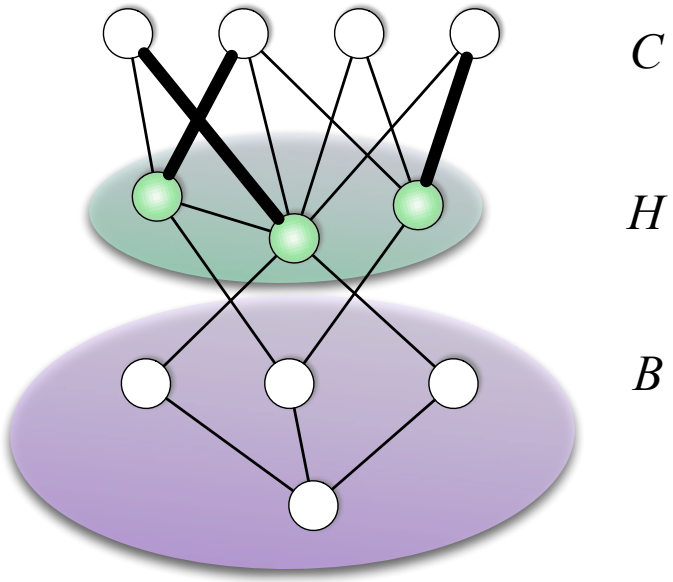


Crown rule for Vertex Cover:
The matching needs to be covered and we can assume that it is covered by H (makes no sense to use vertices of C)

Crown Reduction

Definition: A **crown decomposition** is a partition $C \cup H \cup B$ of the vertices such that

- C is an independent set,
- there is no edge between C and B,
- there is a matching between C and H that covers H.



Crown rule for Vertex Cover:
The matching needs to be covered and we can assume that it is covered by H (makes no sense to use vertices of C)
$\Rightarrow (G \setminus (H \cup C), k - |H|)$.

Crown Reduction

Key lemma:

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$, $\Rightarrow$ No solution!

Crown Reduction

Key lemma:

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$, $\Rightarrow$ No solution!
- find a crown decomposition, $\Rightarrow$ Reduce!

Crown Reduction

Key lemma:

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$, $\Rightarrow$ No solution!
- find a crown decomposition, $\Rightarrow$ Reduce!
- or conclude that the graph has at most $3k$ vertices.

Crown Reduction

Key lemma:

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$, $\Rightarrow$ No solution!
- find a crown decomposition, $\Rightarrow$ Reduce!
- or conclude that the graph has at most $3k$ vertices.
- $\Rightarrow$ $3k$ vertex kernel!

Crown Reduction

Key lemma:

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$, $\Rightarrow$ No solution!
- find a crown decomposition, $\Rightarrow$ Reduce!
- or conclude that the graph has at most $3k$ vertices.
- $\Rightarrow$ $3k$ vertex kernel!

This gives a $3k$ vertex kernel for Vertex Cover.

Proof

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$,
- find a crown decomposition,
- or conclude that the graph has at most $3k$ vertices.

For the proof, we need the classical Kőnig's Theorem.

$\tau(G)$: size of the minimum vertex cover
$\nu(G)$: size of the maximum matching (independent set of edges)

Theorem: [Kőnig, 1931] If G is **bipartite,** then

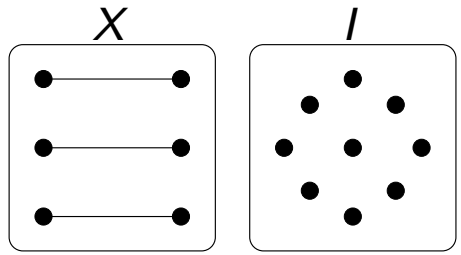
$$\tau(G) = \nu(G)$$

Proof

Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$,
- find a crown decomposition,
- or conclude that the graph has at most $3k$ vertices.

Proof:
Find (greedily) a maximal matching; if its size is at least $k+1$, then we are done. The rest of the graph is an independent set I.



Proof

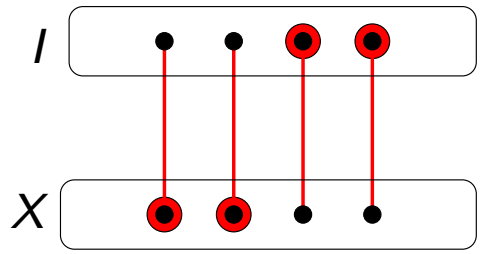
Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$,
- find a crown decomposition,
- or conclude that the graph has at most $3k$ vertices.

Proof:

Find (greedily) a maximal matching; if its size is at least $k+1$, then we are done. The rest of the graph is an independent set I.

Find a maximum matching/minimum vertex cover in the bipartite graph between X and I.



Proof

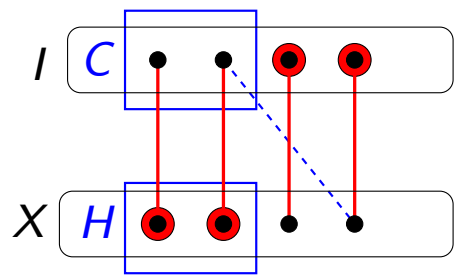
Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$,
- find a crown decomposition,
- or conclude that the graph has at most $3k$ vertices.

Proof:

Case 1: The minimum vertex cover contains at least one vertex of X

$\Rightarrow$ There is a crown decomposition.



Proof

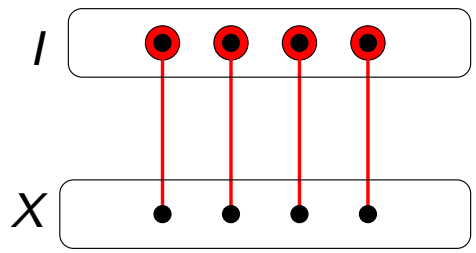
Lemma: Given a graph G without isolated vertices and an integer k, in polynomial time we can either

- find a matching of size $k+1$,
- find a crown decomposition,
- or conclude that the graph has at most $3k$ vertices.

Proof:

Case 1: The minimum vertex cover contains at least one vertex of X
$\Rightarrow$ There is a crown decomposition.

Case 2: The minimum vertex cover contains only vertices of I $\Rightarrow$ It contains every vertex of I
$\Rightarrow$ There are at most $2k + k = 3k$ vertices.



Conclusion

- Kernelization can be thought of as a polynomial-time preprocessing before attacking the problem with whatever method we have. "It does no harm" to try kernelization.
- Some kernelizations use lots of simple reduction rules and require a complicated analysis to bound the kernel size...
- ... while other kernelizations are based on surprising nice tricks and deep mathematical ideas.
- Possibility to prove lower bounds.

Further reading


Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin.
On problems without polynomial kernels.
J. Comput. Syst. Sci., 75(8):423–434, 2009.


Jiong Guo and Rolf Niedermeier.
Invitation to data reduction and problem kernelization.
SIGACT News, 38(1):31–45, 2007.


Neeldhara Misra, Venkatesh Raman, and Saket Saurabh.
Lower bounds on kernelization.
Discrete Optim., 8(1):110–128, 2011.