An introduction to fixed parameter tractability and kernelization

Hans L. Bodlaender

Universiteit Utrecht

On the topic

- Fixed parameter tractability: recent direction in algorithm research
- Kernelization: offspring – allows mathematical analysis of preprocessing for problems
- This talk: informal introduction to central notions
  - Simple examples
  - Some definitions, proofs, algorithms
  - Not much “uncertainty” / ECSQARU problems discussed in this talk…

Schedule

1. Fixed parameter tractability
2. Hardness
3. Kernelization
4. Kernel lower bounds
5. Conclusions

1

Introduction

Parameterized complexity:

What is it about

Universiteit Utrecht

Fixed Parameter Complexity

- Many problems have a *parameter*
- Many applications have this parameter to be *small*
  - E.g.: facility location with small number of facilities (place *k* hospitals on a large map)
  - Structural parameter of input that is likely to be small
- Sometimes, faster / better algorithms are possible

Parameterized problem

- Problem with two argument input:
  - Given: Some information x, integer k, …
  - Parameter: k
  - Question: Q(x,k)?
- Many examples ...

Examples of parameterized problems (1)

Graph Coloring

Given: Graph G, integer k

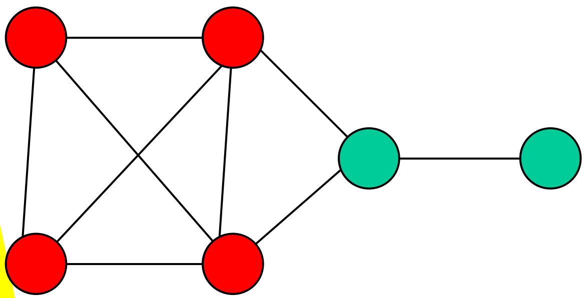
Parameter: k

Question: Is there a vertex coloring of G with k colors? (I.e., $c: V \rightarrow \{1, 2, \ldots, k\}$ with for all $\{v,w\} \in E$: $c(v) \neq c(w)$?)

- NP-complete, even when $k=3$.

Clique

- Subset W of the vertices in a graph, such that each pair has an edge



Examples of parameterized problems (2)

Clique

Given: Graph G, integer k

Parameter: k

Question: Is there a clique in G of size at least k?

- Solvable in $\mathbf{O}(\boldsymbol{n}^{\boldsymbol{k}})$ time with simple algorithm. Complicated algorithm gives $O(n^{2k/3})$. Seems to require $\Omega(n^{f(k)})$ time…

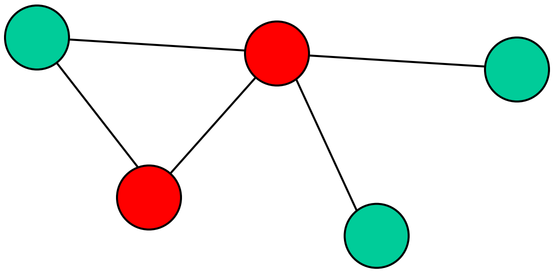
Simple $O(n^k)$ algorithm

- More or less like this:
  - For each set S of k vertices in G:
    - If S is a clique, then return yes
  - (If none returned yes:) return no

- ... hardly anything better known ...

Vertex cover

- Set of vertices $W \subseteq V$ with for all $\{x,y\} \in E$: $x \in W$ or $y \in W$.
- Vertex Cover problem:
  - Given G, find vertex cover of minimum size



Examples of parameterized problems (3)

Vertex cover

Given: Graph G, integer k

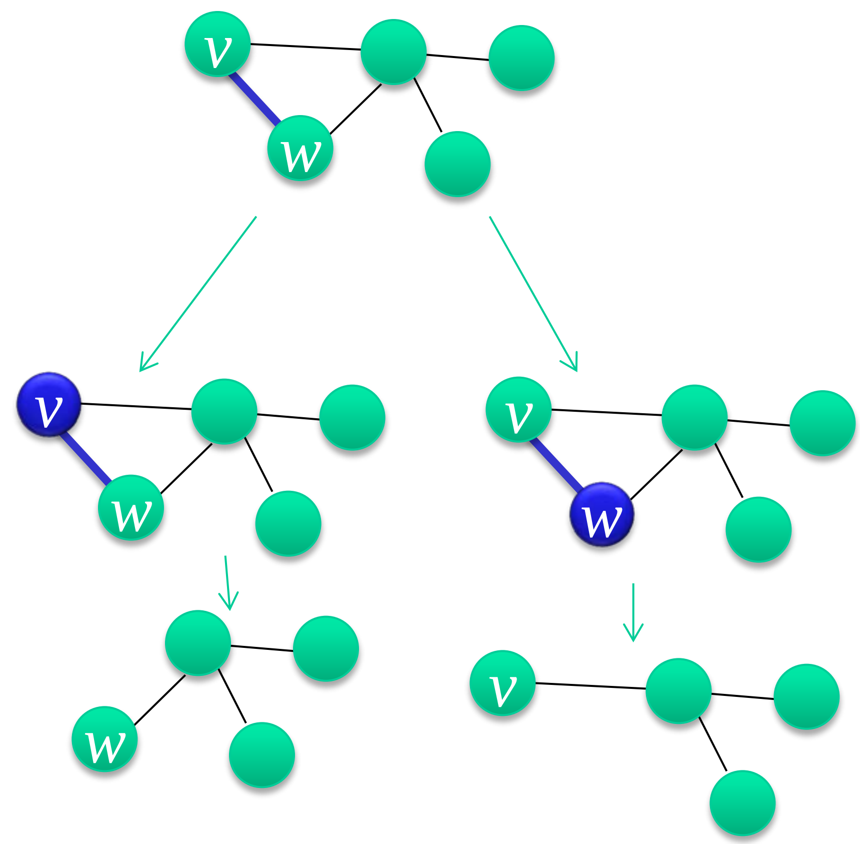
Parameter: k

Question: Is there a vertex cover of G of size at most k?

- Solvable in $O(2^k (n+m))$ time

Idea for algorithm

- Take an edge $\{v,w\}$
- In each solution S, we have v or w (or both)
- If we take v, then this is similar to looking at the graph obtained by removing v and all its edges



Vertex Cover in $O(2^k (n+m))$ time

- Recursive algorithm
- VC(G, k)
  - If G has no edges: return yes
  - If k == 0: return no
  - Choose an edge $e = \{v,w\}$
  - Let G' be obtained from G by removing v and all its edges
  - Let G'' be obtained from G by removing w and all its edges
  - Return VC(G',k-1) or VC(G'',k-1)

Three types of complexity

- When the parameter is fixed
  - Still NP-complete (k-coloring, take k=3)
  - $O(f(k)\, n^c)$
  - $O(n^{f(k)})$

Fixed parameter complexity theory

- To distinguish between behavior:
  - $O(f(k) * n^c)$
  - $\Omega(n^{f(k)})$
- Proposed by Downey and Fellows.

Parameterized problems

- Instances of the form (x,k)
  - I.e., we have a *second parameter*
- Decision problem (subset of $\{0,1\}^* \times \mathbf{N}$)

- Notation: k is the parameter, n measures size of x

Fixed parameter tractable problems

- FPT is the class of problems with an algorithm that solves instances of the form (x,k) in time $p(|x|)*f(k)$, for polynomial p and some function f.
  - E.g. $O(3^k n^2)$, $O(k!\ n)$, ...

Hard problems

- Complexity classes
  - W[1] $\subseteq$ W[2] $\subseteq$ … W[*i*] $\subseteq$ … W[P]
  - Defined in terms of *Boolean circuits*
  - Problems hard for W[1] or larger class are assumed not to be in FPT
    - Compare with P / NP

Examples of hard problems

- Clique and Independent Set are W[1]-complete
- Dominating Set is W[2]-complete
- Version of Satisfiability is W[1]-complete
  - Given: set of clauses, k
  - Parameter: k
  - Question: can we set (at most) k variables to **true**, and al others to **false**, and make all clauses true?

So what is parameterized complexity about?

- Given a parameterized problem
- Establish that it is in FPT
  - And then design an algorithm that is as fast as possible
- *Or* show that it is hard for W[1] or "higher"
  - Try to find a polynomial time algorithm for fixed parameter
- *Or* even show that it is NP-complete for fixed parameters
  - Solve it with different techniques (exact or approximation)

FPT techniques

- Several algorithmic techniques to show that problems are in FPT
  - Branching
  - Dynamic programming
    - Exploiting structures like tree decompositions (clique trees, junction trees); linear structure of problem instances ...
  - Advanced, specialized techniques:
    - Iterative improvement
    - Color coding
    - ...

Closest string

- Given: k strings $s_1, \ldots, s_k$ each of length L, integer d
- Parameter: d
- Question: is there a string s with Hamming distance at most d to each of $s_1, \ldots, s_k$

- Application in molecular biology
- Here: FPT algorithm
- (Gramm and Niedermeier, 2002)

Subproblems

- Subproblems have form
  - Candidate string *s*
  - Additional parameter *r*
  - We look for a solution to original problem, with additional condition:
    - Hamming distance at most *r* to *s*
- Start with $s = s_1$ and $r=d$ (= original problem)

Branching step

- Choose an s_j with Hamming distance > d to s
- If Hamming distance of s_i to s is larger than $d+r$: *NO*
- For all positions i where s_j differs from s
  - Solve subproblem with
    - s changed at position i to value s_j (j)
    - $r = r - 1$
- Note: we find a solution, if and only one of these subproblems has a solution

Example

- Strings 01112, 02223, 01221, d=3
  - First position in solution will be a 0
  - First subproblem (01112, 3)
  - Creates three subproblems
    - (02113, 2)
    - (01213, 2)
    - (01123, 2)

Time analysis

- Recursion depth d
- At each level, we branch at most at $d + r \leq 2d$ positions
- So, number of recursive steps at most d^{2d+1}
- Each step can be done in polynomial time: $O(kdL)$
- Total time is $O(d^{2d+1} \cdot kdL)$
- Speed up possible by more clever branching and by kernelisation

Technique

- Try to find a branching rule that
  - Decreases the parameter
  - Splits in a bounded number of subcases
    - YES, if and only if YES in at least one subcase

Color coding

- Interesting algorithmic technique to give fast FPT algorithms
- As example:
- Long Path
  - Given: Graph G=(V,E), integer *k*
  - Parameter: *k*
  - Question: is there a simple path in G with at least *k* vertices?

Problem on colored graphs

- Given: graph G=(V,E), for each vertex v a color in $\{1,2, \ldots, k\}$
- Question: Is there a simple path in G with k vertices of different colors?
  – Note: vertices with the same colors may be adjacent.
- Can be solved in $O(2^k (nm))$ time using dynamic programming
- Used as subroutine…

DP

We skip this slide

- Tabulate:
  - (S,*v*): S is a set of colors, *v* a vertex, such that there is a path using vertices with colors in S, and ending in *v*
  - Using Dynamic Programming, we can tabulate all such pairs, and thus decide if the requested path exists

A randomized variant

- For each vertex v, ***guess a color*** in $\{1, 2, \ldots, k\}$
- Check if there is a path of length k with only vertices with different colors
- Note:
  - If there is a path of length k, we find one with positive chance ($2^k / k!$)
  - We can do this check in $O(2^k nm)$ time
  - Repeat the check many times to get good probability for finding the path

From randomized to deterministic

- Randomized algorithm:
  - Repeat many times:
    - Guess colors
    - Solve DP; if YES, then return YES
  - Return NO
- Derandomization is possible with *k-perfect family of hash functions* (replacing guesses)...

4

Hardness proofs

Remember Cook/Levin theorem

- NP-completeness helps to distinguish between decision problems for which we have a polynomial time algorithm, and those for which we expect no such algorithm exists
- NP-hard; NP-completeness; reductions
- Cook/Levin theorem: `first' NP-complete problem; used to prove others to be NP-complete
- Similar theory for parameterized problems by Downey and Fellows

Classes

- FPT $\subseteq$ W[1] $\subseteq$ W[2] $\subseteq$ W[3] $\subseteq$ … $\subseteq$ W[i] $\subseteq$ … $\subseteq$ W[P]
- Theoretical reasons to *believe* that hierarchy is strict
- Theorem: If FPT = W[1], then the Exponential Time Hypothesis does not hold
- ETH (Impagliazzo et al., 1999): There is a δ such that 3-Satisfiability cannot be solved in $O(2^{\delta n})$ time

Scheme

- Define a notion of *reduction*
- From the notion of reduction, we get *hardness* and *completeness*
- Generic hard/complete problems + reduction give new hard/complete problems

Parameterized *m*-reduction

- Let L, L' be parameterized problems.
- A *standard parameterized m-reduction* transforms an input (I,k) of L to an input (f(I,k), g(k)) of L'
  - L((I,k)) if and only if L'((f(I,k), g(k))
  - f uses time p(|I|)* h(k) for a polynomial p, and some function h
- Note: time may be exponential or worse in k
- Note: the parameter only depends on parameter, not on rest of the input

A Complete Problem

- Classes W[1], … are defined in terms of circuits (definition skipped here)
- Short Turing Machine Acceptance
  - Given: A non-deterministic Turing machine M, input x, integer k
  - Parameter: k
  - Question: Does M accept x in a computation with at most k steps?
- Short Turing Machine Acceptance is W[1]-complete (compare Cook)
- Note: easily solvable in $O(n^{k+c})$ time

More complete problems for W[1]

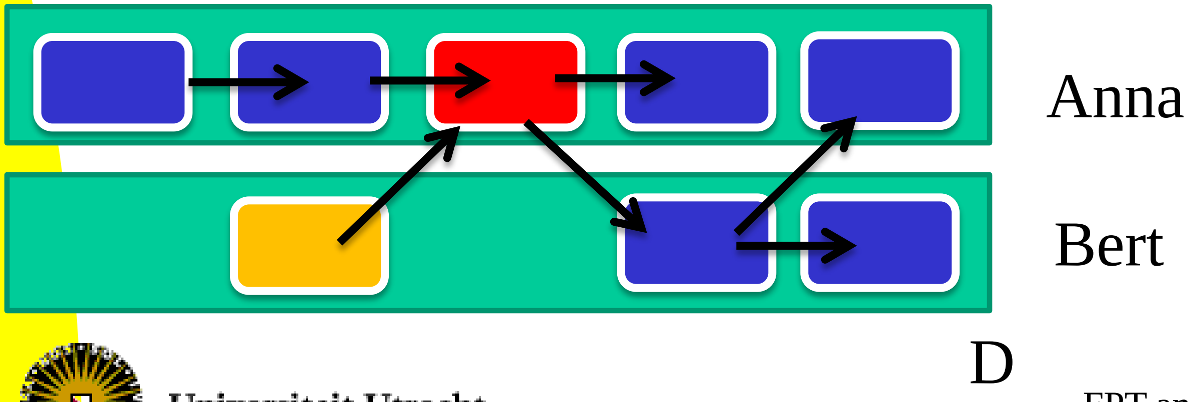
- Weighted q-CNF Satisfiability
  - Given: Boolean formula in CNF, such that each clause has at most q literals, integer k
  - Parameter: k
  - Question: Can we satisfy the formula by making at most k literals true?
- For each fixed $q > 1$, Weighted q-CNF Satisfiability is complete for W[1].

Hard problems

- Independent Set, Clique: W[1]-complete
- Dominating Set: W[2]-complete
- Longest Common Subsequence III: W[1]-complete (complex reduction to Clique)
  - Given: set of k strings $S^1, \ldots, S^k$, integer m
  - Parameter: k, m
  - Question: is there a string S of length m that is a subsequence of each string S^i, $1 \leq i \leq k$?

Example reduction

- K people have to do *n* tasks. Each task costs 1 hour. Some tasks have to be done before some other tasks, and there is a deadline D.
- Can we finish all tasks before D?

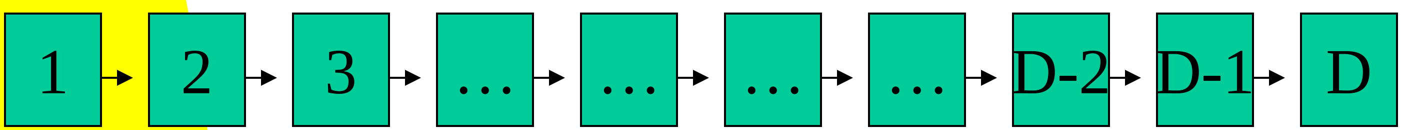


Formal problem

- Precedence constrained K-processor scheduling
  - Instance: set of tasks T, each taking 1 unit of time, partial order < on tasks, deadline D, number of people that can carry out tasks K
  - Parameter: K
  - Question: can we carry out the tasks by K people, such that
    - If task1 < task2, then task1 is carried out before task2
    - At most one task per time step per person
    - All tasks finished at most at time D

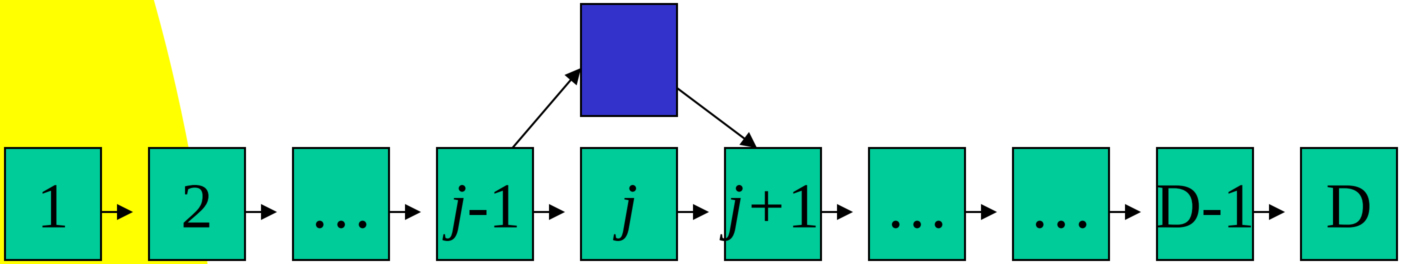
Transform from Dominating Set

- Let G=(V,E), k be instance of DS
- Write $n = |V|$, $c = n^2$, $D = knc + 2n$.
- Take the following tasks and precedences:
- Floor: D tasks in “series”:



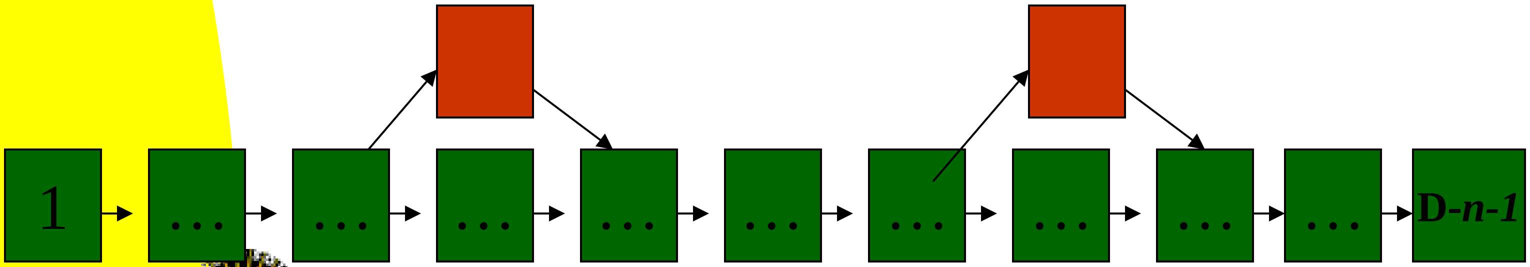
Floor gadgets

- For all j of the form $j = n$-1+ $ac + bn$ ($0 \leq a < kn$, $1 \leq b \leq n$), take a task that must happen on time j (parallel to the jth floor vertex)



Selector paths

- We take k paths of length D-n+1
- Each models a vertex from the dominating set
- To some vertices on the path, we also take parallel vertices:
  - If $\{v_i, v_j\} \notin E$, and $i \neq j$, then place a vertex parallel to the n-1+ac+in-j^{th} vertex for all a, $0 \leq a < kn$



Lemma and Theorem

- Lemma: we can schedule this set of tasks with deadline D and *2k* processors, if and only if G has a dominating set of size at most k
- Theorem: Precedence constrained k-processor scheduling is W[2]-hard
- Note: size of instance must depend in polynomial way on size of G (and hence on $k < |V|$)
- It is allowed to use transformations where new parameter is exponential in old parameter

About these hardness proofs

- Fixed parameter proofs: method of showing that a problem *probably* has no FPT-algorithms
- Often complicated proof ☹
- But not always ☺

4

Kernelisation

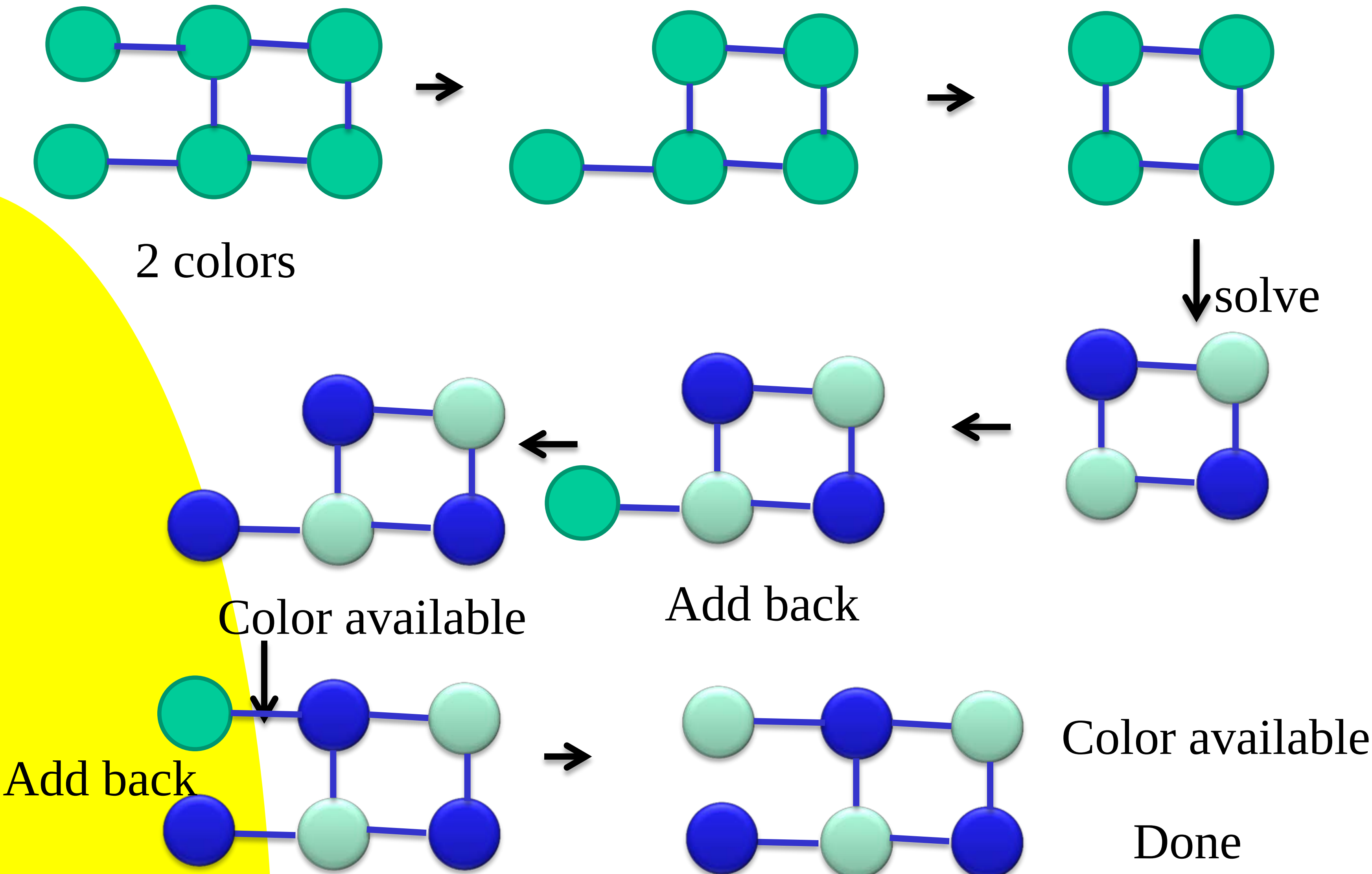
Universiteit Utrecht

Preprocessing

- Useful technique for solving problems:
  - Preprocess: change instance *x* to equivalent but smaller instance *y*
  - Solve the problem on *y* obtaining solution *s'*
  - Translate *s'* back to a solution *s* for *x*
- Used very frequently (e.g., CPLEX, sat-solvers, etc. etc.)

Example 0

- Graph coloring:
  - Given: Graph G, number of colors *c*
  - Question: can we vertex color G with at most *c* colors?
- Heuristic preprocessing: remove vertices with at most *c*-1 neigbors, while they exist
- Undoing preprocessing:
  - Suppose we have a coloring of the reduced graph
  - Add the removed vertices back in reverse order. When we add a vertex, it has at most c-1 neighbors, so we can color it now.


2 colors
solve
Add back
Color available
Add back
Color available
Done

The area of Kernelization

- Central question: what can we say about the size of a resulting instance?

- What we cannot hope for:
  - An algorithm that always reduces the size of the input to a smaller equivalent one?
  - Why not ... ?

Kernelization

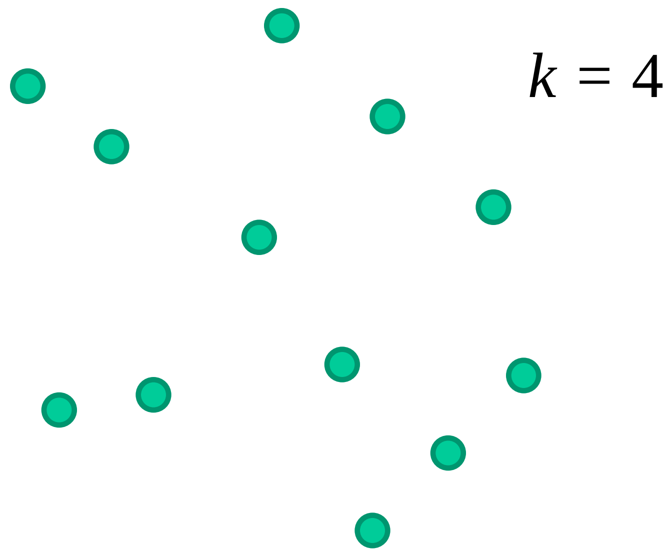
- Preprocessing that is:
  - **Safe**: the answer to the question does not change
  - Guarantee on size of resulting input as function of a parameter
  - Fast (polynomial time)
- We look at decision problems (answer yes or no)

Kernelization

- Preprocessing rules reduce starting instance to one of size f(k)
  - Should work in polynomial time
- Then use any algorithm to solve problem on kernel
- Time will be p(n) + g(f(k))

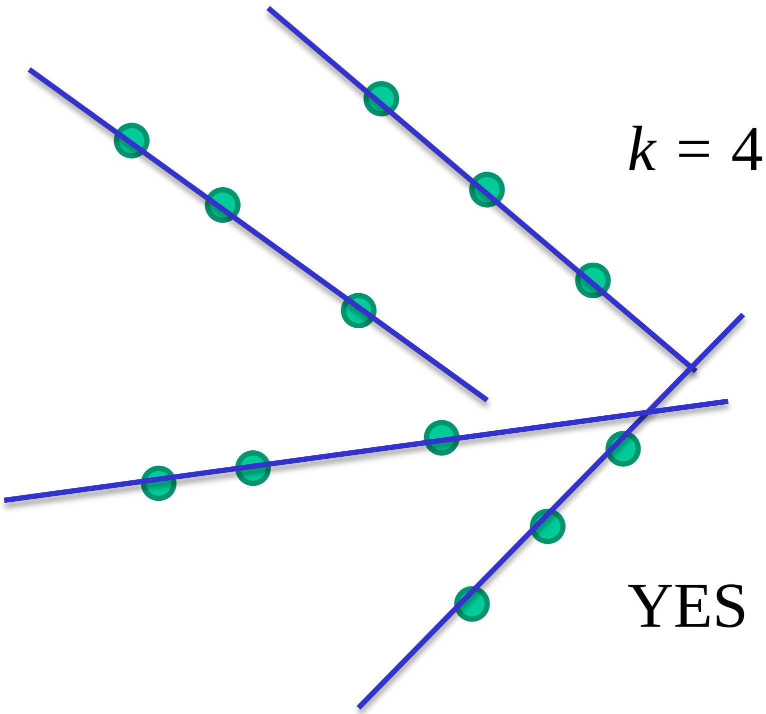
Example problem Point-Line-Cover

- Given: Set S of points in the plane, integer *k*
- Parameter: *k*
- Question: are there *k* straight lines that hit all the points



Example problem Point-Line-Cover

- Given: Set S of points in the plane, integer k
- Parameter: k
- Question: are there k straight lines that hit all the points?

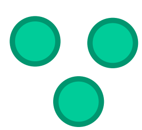


Rule 1

- Observation: if we have a line that hits k+1 points, we have to take it
  - Otherwise ...
- Rule 1: If we have a line that hits k+1 or more points, then
  - Remove the points hit by the line
  - Set $k = k - 1$

Rule 2

- Observation: suppose no line hits more than k points. If we have more than k^2 points, we need more than k lines
- Rule 2: If we cannot apply Rule 1, and we have more than k^2 points then say NO
  - Formally: change instance to trivial NO-instance



1

Kernel for Point-Line-Cover

Algorithm:

- While Rule 1 is possible, apply it
- If Rule 2 is possible, apply it

- Easy: polynomial time
- Trivial: afterwards, we have at most k^2 points

Maximum Satisfiability

- Given: Boolean formula in conjunctive normal form; integer k
- Parameter: k
- Question: Is there a truth assignment that satisfies at least k clauses?
- Denote: number of clauses: C

Skip this part

Universiteit Utrecht

Reducing the number of clauses

- If $C \geq 2k$, then answer is YES
  - Look at arbitrary truth assignment, and truth assignment where we flip each value
  - Each clause is satisfied in one of these two assignment
  - So, one assignment satisfies at least half of all clauses

Bounding number of long clauses

- Long clause: has at least *k* literals
- Short clause: has at most *k*-1 literals
- Let L be number of long clauses
- If L ≥ *k*: answer is YES
  - Select in each long clause a literal, whose complement is not yet selected
  - Set these all to true
  - All long clauses are satisfied

Reducing to only short clauses

- If less than *k* long clauses
  - Make new instance, with only the short clauses and *k* set to *k*-L
  - There is a truth assignment that satisfies at least *k*-*L* short clauses, if and only if there is a truth assignment that satisfies at least *k* clauses
    - =>: choose for each satisfied short clause a variable that makes the clause true. We may change all other variables, and can choose for each long clause another variable that makes it true
    - <=: trivial

An $O(k^2)$ kernel for Maximum Satisfiability

- If at least $2k$ clauses: return YES
- If at least k long clauses: return YES
- Else: remove all L long clauses, and set $k=k$-L

Formal definition of kernelisation

- Let P be a parameterized problem. (Each input of the form (I,k).) A *reduction to a problem kernel* is an algorithm, that transforms A inputs of P to inputs of P, such that
  - P((I,k)), if and only if P(A(I,k)) for all (I,k)
  - If A(I,k) = (I',k'), then $k' \leq f(k)$, and $|I'| \leq g(k)$ for some functions f, g
  - A uses time, polynomial in |I| and k

A theorem with a strange proof

- Theorem (folklore): Let Q be a decidable parameterized problem. The following are equivalent:

1. Q belongs to FPT, i.e., has an algorithm with time $O(f(k)n^c)$ for fixed c
2. Q has a (reduction to a problem) kernel

Proof part 1

- Suppose Q has a kernel. Then this is an FPT algorithm:
  - Given: instance x, parameter k
  - Build the kernel y, k' (has size f(k))
  - Run any algorithm to decide on y, k'
- Running time is $p(|x|) + g(f(k))$ for polynomial p and some function g
  - $p(|x|)$ for making kernel
  - $g(f(k))$ for solving kernel

Proof part 2

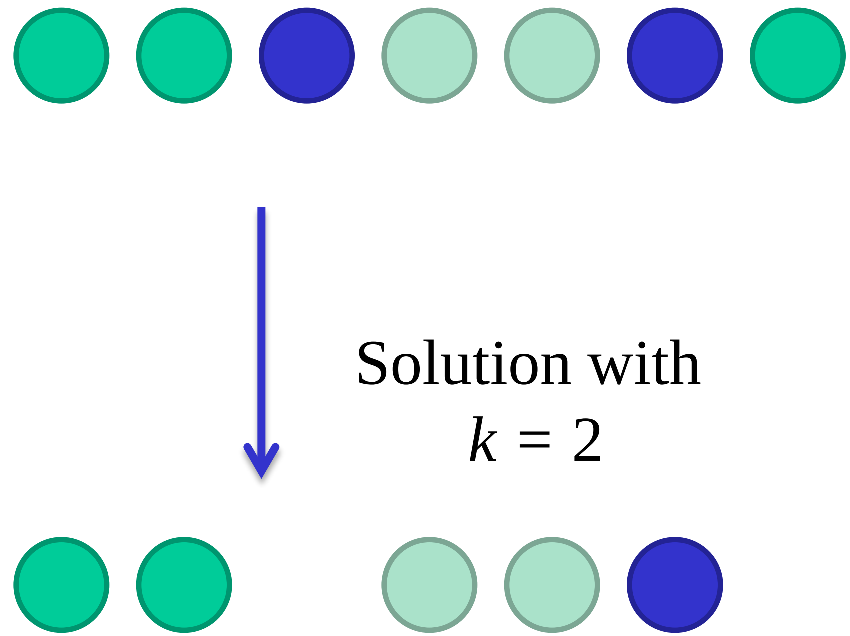
- 😐
- If P is in FPT, P has a (perhaps trivial) reduction to a problem kernel
  - Given: instance x, parameter k
  - Suppose we have an $f(k)\, n^c$ algorithm
  - If $|x| > f(k)$, solve problem exactly: this is $O(n^{c+1})$ time
    - Formality: take small yes or no-instance afterwards
  - Otherwise, output x, k (i.e., do nothing)
    - We have that $|x| <= f(k)$.

Implications of the theorem

- Positive:
  - Technique to obtain FPT-algorithms:
    - Make small kernel.
    - Algorithm on resulting small instance.
- Negative:
  - If we have evidence that there exists no FPT-algorithm, we also have evidence that there exists no kernel.

Another kernel example

- Convex colored marbles
  - Real application in computational biology
  - Given: sequence of colored marbles, integer k
  - Parameter: k
  - Question: can we remove at most k marbles such that for each color, all marbles with that color are consecutive?

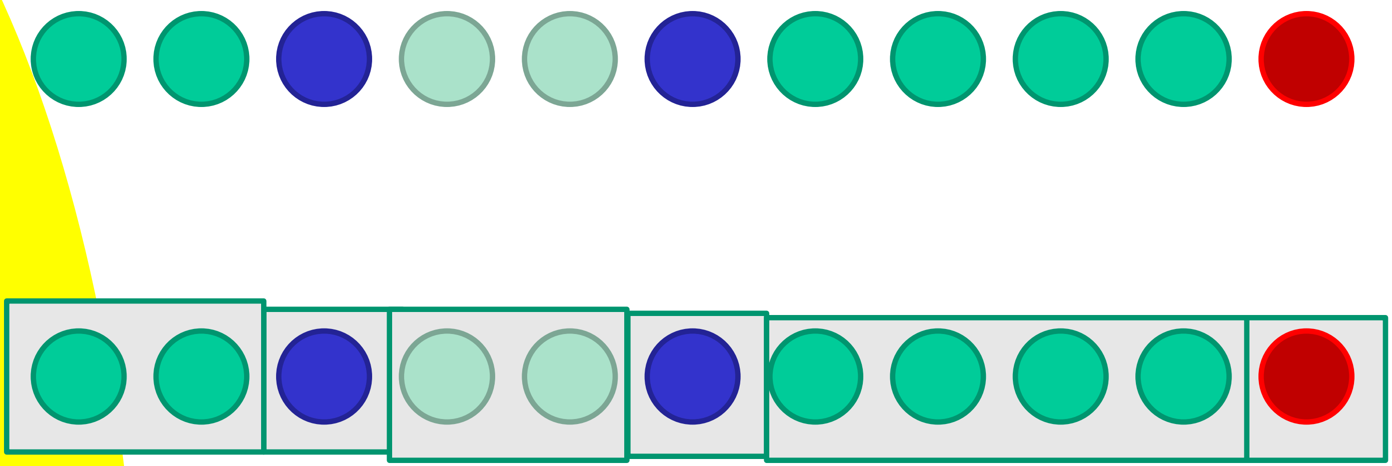


Algorithm scheme

- Some safe rules:
  - Do not change answer to the problem
  - Simplify the instance
- Apply the rules while possible
- Argument that resulting instance has bounded size
- Plan: build rules that limit some aspect of the input

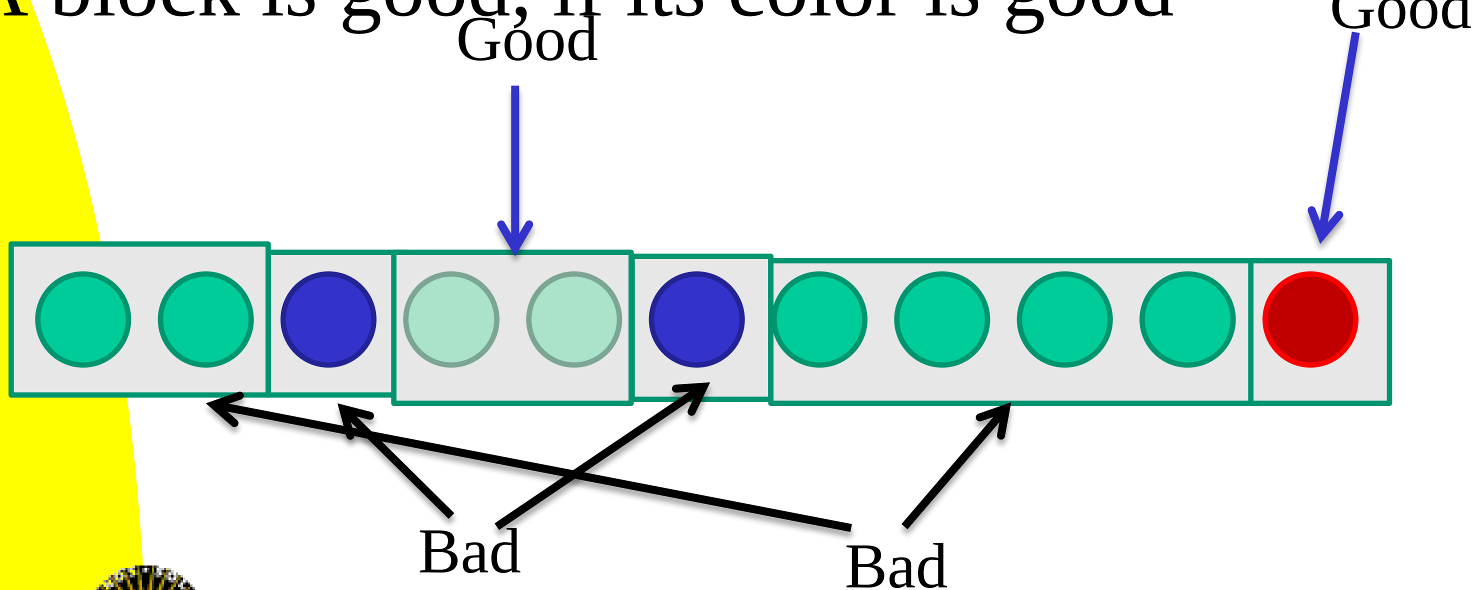
Blocks

- A block is a maximum consecutive part of similarly colored marbles



Good colors and bad colors

- A color is *good*, if there is only one block with this color, otherwise it is bad
- A block is good, if its color is good

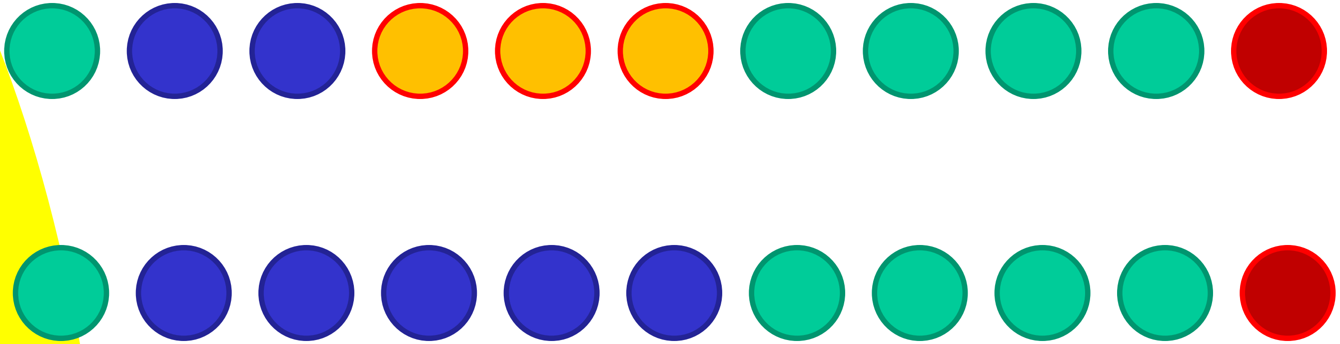


Reducing number of blocks of bad colors

- Observation: each removal can reduce the number of bad blocks by at most 4
  - The removed marble
  - The two neighboring blocks could become one
- Rule 1: If there are more than 4k blocks with a bad color, say NO

Rule 2

- If we have two consecutive good blocks, give them the same color



Counting

- We have at most 4k bad blocks, and each good block is between bad blocks: at most 4k+1 good blocks, and at most 8k+1 blocks
- We need some way to bound the size of blocks ...

Bounding the size of blocks

- Rule 3: If a block has more than k+1 marbles, change its size to k+1
  – Why correct?

- Resulting algorithm:
  – Apply rules while possible
- Kernel size: at most (8k+1)(k+1) marbles

Many small kernels exist

- Graph problems: Feedback vertex set, vertex cover, many problems on planar graphs, ...
- Logic: can we satisfy at least *k* clauses of a Satisfiability instance in CNF, ...
- ...

Negative results

- Recall:

Theorem If W[1] = FPT, then the *Exponential Time Hypothesis is not valid*.

Corollary A parameterized problem that is W[1]-hard has no kernel, unless the ETH does not hold.

Many W[1]-hard problems

- Many problems are W[1]-hard, e.g.: Clique, Independent Set, Dominating Set, …
- No kernels for these, unless W[1] = FPT and hence the Exponential Time Hypothesis fails.

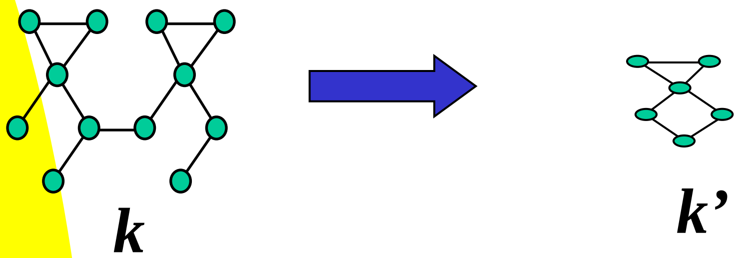
Problems with large kernels

- For many problems in FPT, we do not know small kernels.
- Consider:

  Long Path
  - Given: Graph G=(V,E), integer *k*.
  - Question: Does G have a simple path of length at least *k*?
  - Parameter: *k*.
- Is in FPT, but all known kernels have size exponential in *k*…

Does Long Path have a kernel of polynomial size? Maybe not…

- Suppose we have a polynomial kernel, say with k^c bits size.



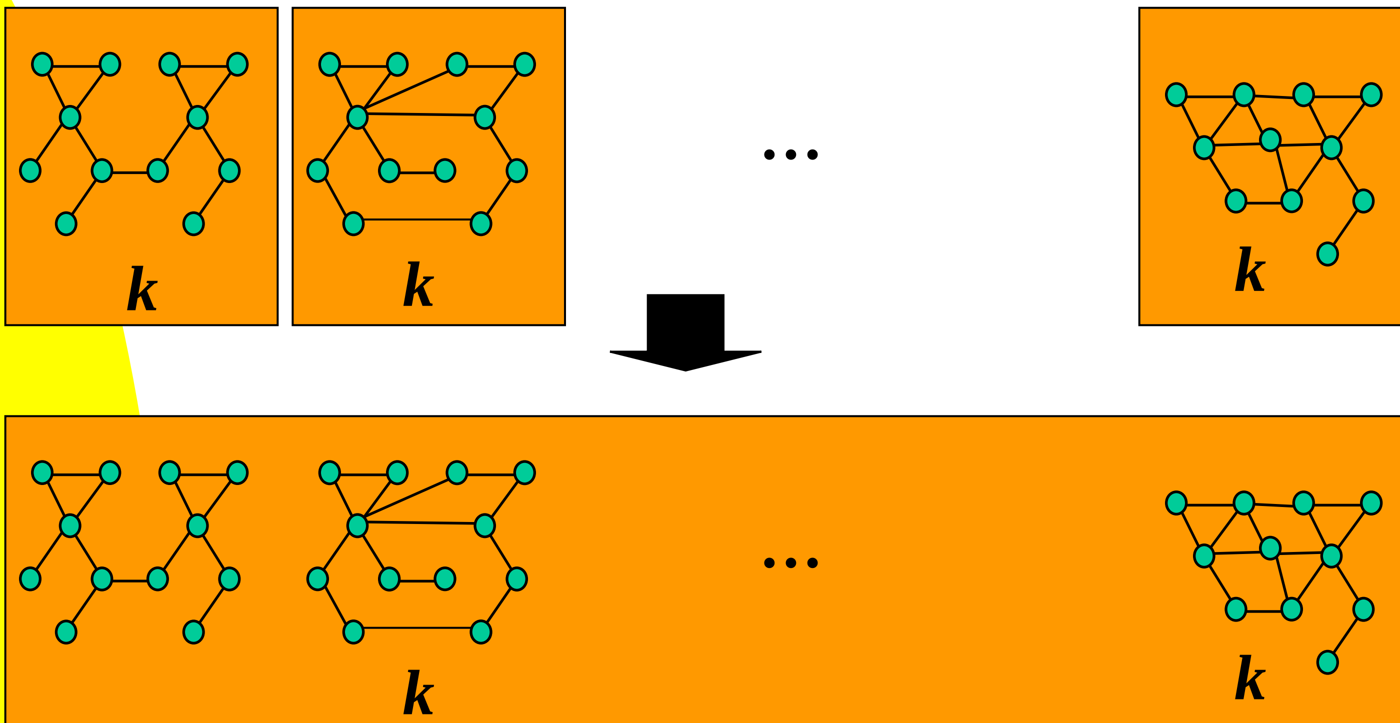
Size bounded by k^c

Long path continued

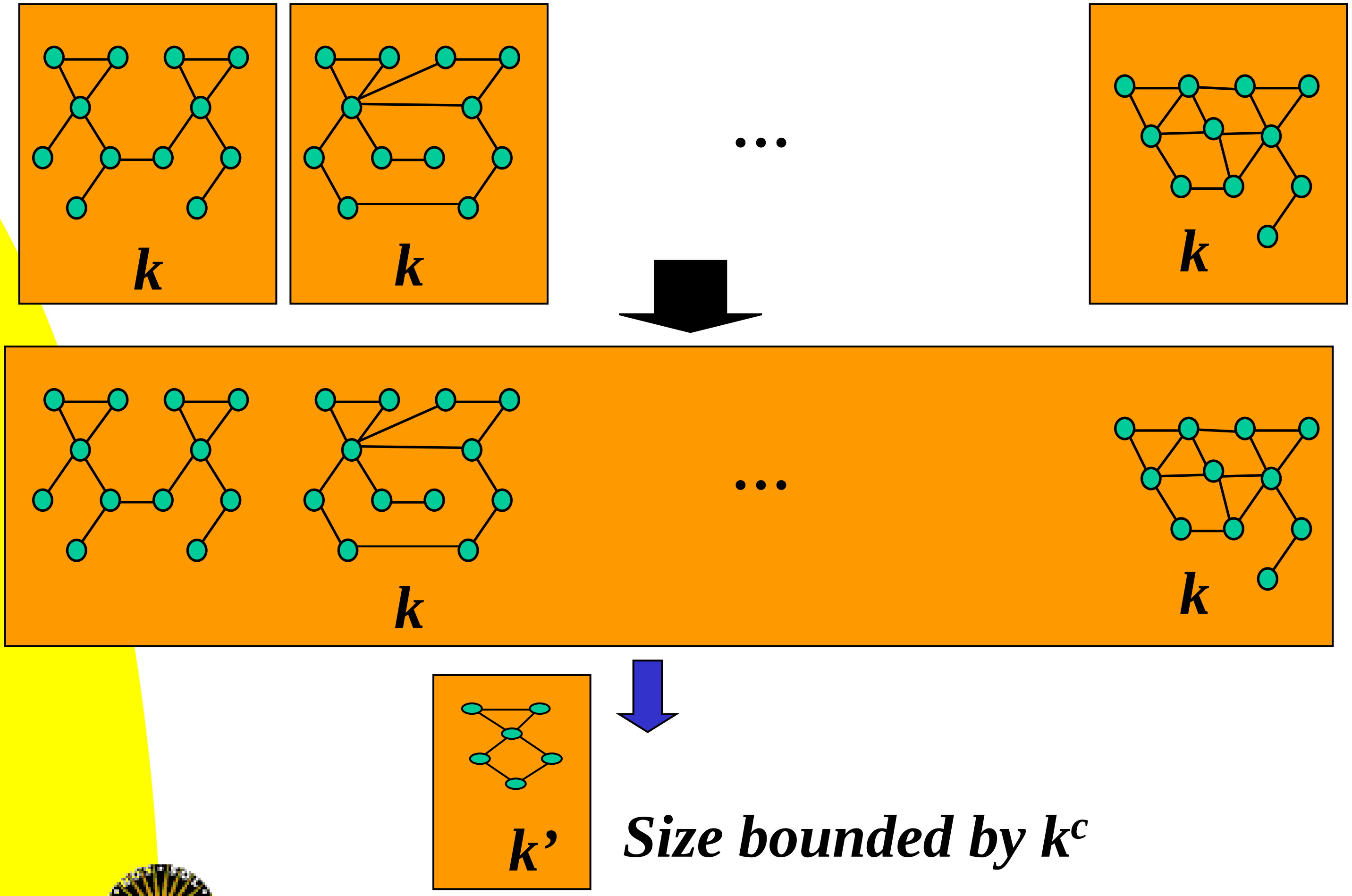
- Now, suppose we have a series of inputs to long path, say all with the same parameter: (G_1,k), (G_2,k), …, (G_r,k).

Take the disjoint union

- $G_1 \cup G_2 \cup \ldots \cup G_r$ has a simple path of length k, if and only if there exists a graph G_i that has a path of length k.



And now, apply the kernel to the union



Size bounded by k^c

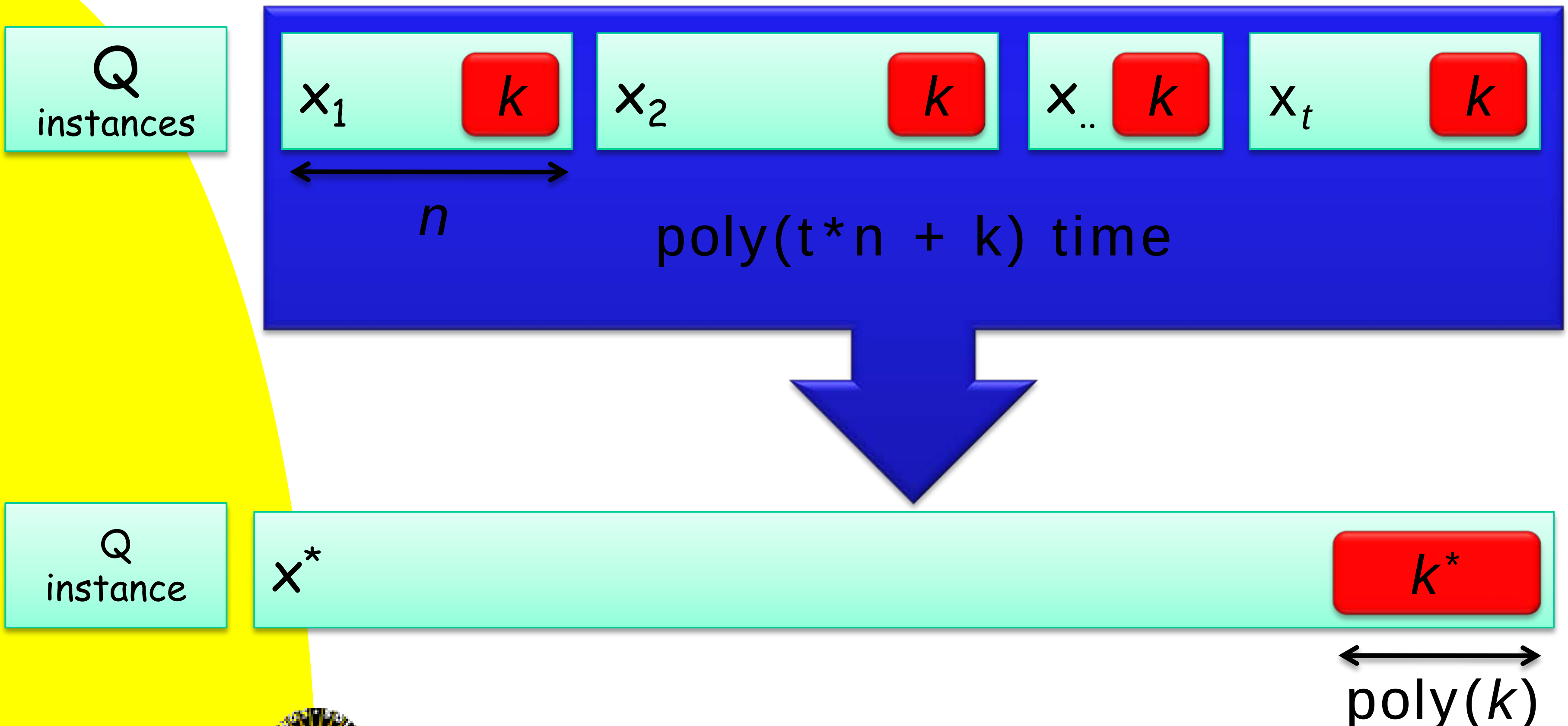
What happened?

- We have many (say $r = k^{2c}$) instances of Long Path, and transform it to one instance of size $< k^c$.
- *Intuition*: this cannot be possible without solving some of the instances, as we have fewer bits left than we had instances to start with…
- Theory (next) formalizes this idea.

(Or-)Compositionality

- A parameterized problem Q is *or-compositional*, if there is an algorithm that
  - Receives as input a series of inputs to Q, all with the same parameter $(I_1,k), \ldots, (I_r,k)$;
  - Uses polynomial time;
  - Outputs one input (I',k') to Q;
  - k' bounded by polynomial in k;
  - $(I',k') \in Q$ if and only if there exists at least one j with $(I_j,k) \in Q$.

Or-composition



Universiteit Utrecht

Compositionality gives lowerbounds for kernels

Theorem (B, Downey, Fellows, Hermelin + Fortnow, Santhanam, 2008**)**
Let P be a parameterized problem that is

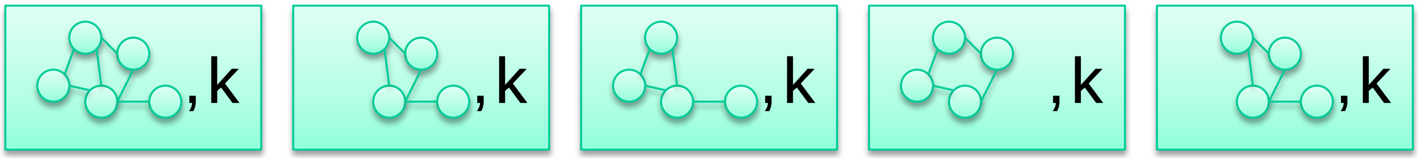
- Or-compositional, and
- "Unparameterized form" is NP-complete.

Then P has no polynomial kernel unless NP $\subseteq$ coNP/poly.

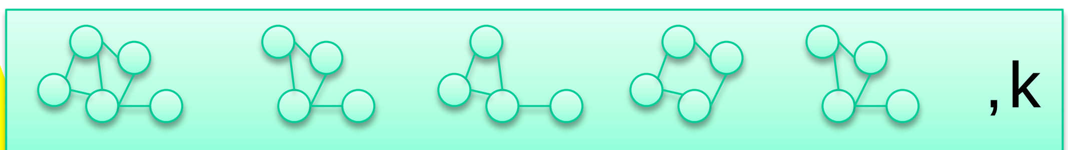
- Variant for and-compositionality also exists, with recent (2012) result by Drucker

Application to Long Path

- Input: t instances of Longest Path.



- Take disjoint union, output as (G’, k).



- G’ has a path of length k ⇔ some G_i has a path of length k.
- Output parameter trivially bounded in poly(k).

Long Path does not admit a polynomial kernel unless NP⊆coNP/poly

Additional techniques (1)

- Polynomial parameter transformations (several authors): transform an argument that problem X does not have a polynomial kernel to an argument that problem Y does not have a polynomial kernel.
- Chen et al. (2009): no kernels of size $\mathbf{k^c\ n^{1-\varepsilon}}$ (unless NP $\subseteq$ coNP/poly).
- Cross-compositions (B, Jansen, Kratsch, 2010): (composition of instances of problem X into instances of problem Y).
  - Composition of 2^n instances suffices.

Additional techniques (2)

- Dell and van Melkebeek (2010): extend technique to precise lower bounds, e.g.: $\Omega(k^2)$ bits for kernel for Vertex Cover (unless NP $\subseteq$ coNP/poly).
- E.g.: Kratsch (2013, unpublished): there is no kernel for Point-Line-Cover with $O(k^{2-\varepsilon})$ points unless NP $\subseteq$ coNP/poly for $\varepsilon>0$.
- ...

Disjoint cycles

- Disjoint cycles
  - Given: Graph G=(V,E), integer *k*.
  - Question: Does G contain *k* vertex disjoint cycles?
  - Parameter: *k*.
- NP-complete, FPT, but does it has a polynomial kernel??
- Resembles Feedback Vertex Set, but behaves differently!
  - Feedback vertex set
    - Given: Graph G, integer *k*.
    - Question: Is there a set of *k* vertices W such that G-W has no cycle?
    - Parameter: *k*.
  - FVS has O(k^2) kernel (Thomassé)

PPT-transformation

- A polynomial-parameter-time transformation (ppt-transformation) P to Q is an algorithm
  - which takes an instance (x,k) of P as input,
  - uses time polynomial in $|x| + k$,
  - outputs an instance (x', k') of Q with
    - $(x,k) \in P \Leftrightarrow (x', k') \in Q$,
    - k' is polynomial in k.

Theorem: If P has a ppt-transformation to Q, Q is NP-complete, P is in NP, and P has no polynomial kernel, then Q has no polynomial kernel.

Proof

Theorem: If P has a ppt-transformation to Q, Q is NP-complete, P is in NP, and P has no polynomial kernel, then Q has no polynomial kernel.

Proof Suppose Q has a polynomial kernel. Build a polynomial kernel for P as follows:

- Take input (x,k) for P.
- Transform (x,k) to input (y,l) for Q with ppt-transformation.
- Use kernel on (y,l): gives equivalent (y',l') for Q with polynomial size bound on $|y|$.
- NP-completeness gives transformation from Q to P: apply it to (y',l') gives equivalent (x',k') with $|x'|$ polynomially bounded in $|y'|+l'$, which is polynomially bounded in (x,k). ■

Intermediate problem: Disjoint Factors

- Disjoint Factors
  - Given: Integer k, string s on alphabet $\{1, 2, \dots, k\}$.
  - Question: Can we find disjoint substrings $s_1, s_2, \dots, s_k$ in s such that s_i starts and ends with i?
  - Parameter: k
- Disjoint Factors is NP-complete.
- Solvable with Dynamic Programming in $2^k |s|$ time.
- Next: compositionality.

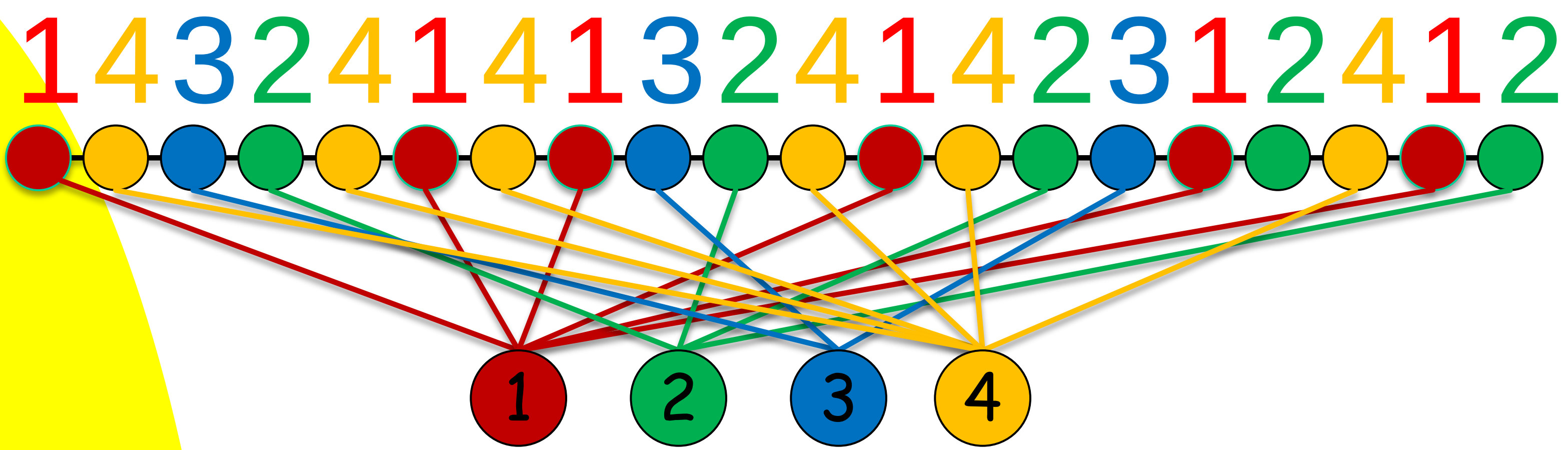
1432414132414231241 2

Disjoint Factors is compositional: proof by example

- Number of instances r can be bounded by 2^k otherwise we can solve them all in polynomial time.
- Take log r new characters, and build new string, like (example for r=4):
  - **b a $\boldsymbol{s_1}$ a $\boldsymbol{s_2}$ a b a $\boldsymbol{s_3}$ a $\boldsymbol{s_4}$ a b**
  - New characters "eat" all but one instance, in which we must then find the other factors:
    - b a s_1 a s_2 a b a $\boldsymbol{s_3}$ a s_3 a b

Corollary: Disjoint Factors has no polynomial kernel unless NP $\subseteq$ coNP/poly.

PPT-transformation from Disjoint Factors to Disjoint Cycles



Disjoint Cycles does not admit a polynomial kernel unless NP⊆coNP/poly

Overview of problem behavior

- O(1) size kernels: problems in P. Ex: Eulerian Graph

  ⇕ NP-completeness (variable parameter)
- Polynomial kernels Shown with algorithm. Ex.: Vertex Cover

  ⇕ compositionality, ppt-transformations, cross-composition
- Kernels, but not polynomial sized. Shown (usually) with FPT-algorithm. Ex: Long Path

  ⇕ W[1]-hardness
- XP: No kernel, polynomial if parameter is bounded. Ex.: Independent Set

  ⇕ NP-completeness (fixed parameter)
- Bad. Example: Graph Coloring is NP-complete for 3 colors

6

Conclusions

Conclusions

- Fixed parameter tractability
  - Tells how to distinguish between $O(f(k)n^c)$ and $O(n^{f(k)})$
  - Practical (and theoretical) algorithms
- Kernelization
  - Analysis of preprocessing
  - Relation with ftp
- Question to you:
  - Do these notions have relevance to your work?