Fixed Parameter Complexity

Algorithms and Networks
Fixed parameter complexity

• Analysis what happens to problem when some parameter is \textit{small}

• Today:
  – Definitions
  – Fixed parameter tractability techniques
    • Branching
    • Kernelisation
    • Other techniques
Motivation

• In many applications, some number may be assumed to be *small*
  – Time of algorithm can be exponential in this small number, but should be polynomial in *usual* size of problem
Example: Can you pick you pick 7 vertices that hit all edges?
Example: Can you pick you pick 7 vertices that hit all edges?
Parameterized graph problem

- **Given**: Graph G, integer $k$, ...
- **Parameter**: $k$
- **Question**: Does G have a ??? of size at least (at most) $k$?
  - Examples: vertex cover, independent set, coloring, …
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 $k=3$. 
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 $O(n^{k+1})$ time with simple algorithm. Complicated algorithm gives $O(n^{2.23k/3})$. Seems to require $\Omega(n^{f(k)})$ time…
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
Fixed parameter complexity theory

• To distinguish between behavior:
  - $O( f(k) \times n^c)$
  - $\Omega( n^{f(k)})$

• Proposed by Downey and Fellows.
Parameterized problems

• Instances of the form \((x, k)\)
  – I.e., we have a second parameter

• Decision problem (subset of \(\{0,1\}^* \times \mathbb{N}\))
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 $f$. 
Hard problems

• Complexity classes
  – $\text{FPT} \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[i] \subseteq \ldots \subseteq W[P]$
  – FPT is ‘easy’, all others ‘hard’
  – Defined in terms of Boolean circuits
  – Problems hard for $W[1]$ or larger classes 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
- This 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 all others to **false**, and make all clauses true?
Techniques for showing fixed parameter tractability

• Branching
• Kernelisation
• Iterative compression
• Other techniques (e.g., treewidth)
A branching algorithm for vertex cover

• Idea:
  – Simple base cases
  – Branch on an edge: one of the endpoints belongs to the vertex cover

• Input: graph $G$ and integer $k$
A branching algorithm for vertex cover

- **Recursive** procedure VC(Graph G, int k)

  - VC(G=(V,E), k)
    - If G has no edges, then return **true**
    - If \( k == 0 \), then return **false**
    - Select an edge \( \{v,w\} \in E \)
    - Compute \( G' = G [V - v] \)
    - Compute \( G'' = G [V - w] \)
    - Return VC(G’,k – 1) or VC(G”’,k – 1)
Analysis of algorithm

• Correctness
  – Either \( v \) or \( w \) must belong to an optimal VC

• Time analysis
  – Branching tree has \( 2^k \) leaves, so \( 2^k \) recursive calls
  – Each recursive call costs \( O(n+m) \) time
  – \( O(2^k (n+m)) \) time: FPT
Cluster editing

- **Instance**: undirected graph $G=(V,E)$, integer $K$
- **Parameter**: $K$
- **Question**: can we make at most $K$ modifications to $G$, such that each connected component is a clique, where each modification is an addition of an edge or the deletion of an edge?
- Models biological question: partition species in families, where available data contains mistakes
- NP-complete. Branching: $O(3^k p(n))$ algorithm
Lemma

- If G has a connected component that is not a clique, then G contains the following subgraph:

- Proof: there are vertices $w$ and $x$ in the connected component that are not adjacent. Take such $w$ and $x$ of minimum distance, which must be 2.
Branching algorithm for Cluster Editing

• If each connected component is a clique:
  – Answer YES

• If $k=0$ and some connected components are not cliques:
  – Answer NO

• Otherwise, there must be vertices $v$, $w$, $x$ with $\{v,w\} \in E$, $\{v,x\} \in E$, and $\{w,x\} \not\in E$
  – Go three times in recursion:
    • Once with $\{v,w\}$ removed and $k = k - 1$
    • Once with $\{v,x\}$ removed and $k = k - 1$
    • Once with $\{w,x\}$ added and $k = k - 1
Analysis branching algorithm

- Correctness by lemma
- Time analysis: branching tree has $3^k$ leaves
More on cluster editing

- Faster branching algorithms exist
- Important applications and practical experiments
- We’ll see more when discussing kernelisation
Max SAT

- Variant of satisfiability, but now we ask: can we satisfy at least $k$ clauses?
- NP-complete
- With $k$ as parameter: FPT
- Branching:
  - Take a variable
  - If it only appears positively, or negatively, then …
  - Otherwise: Branch! What happens with $k$?
Independent Set on Planar Graphs

• **Given**: a planar graph $G=(V,E)$, integer $k$
• **Parameter**: $k$
• **Question**: Does $G$ have an *independent set* of at least $k$ vertices, i.e., a set $W$ of size at least $k$, such that for all $v, w$ in $W$: $\{v, w\}$ is not an edge?
• NP-complete problem
• Here we argue $O(6^k n)$ algorithm (faster is possible)
The red vertices form an independent set
Branching

• Each planar graph has a vertex of degree at most 5
• Take vertex \( v \) of minimum degree, say with neighbors \( w_1, \ldots, w_r, r \) at most 5
• A maximum size independent set contains \( v \) or one of its neighbors
  – Selecting a vertex is equivalent to removing it and its neighbors and decreasing \( k \) by one
• Create at most 6 subproblems, one for each \( x \in \{ v, w_1, \ldots, w_r \} \). In each, we set \( k = k - 1 \), and remove \( x \) and its neighbors
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 none exists, \( s \) is a solution
- If Hamming distance of \( s_j \) to \( s > d+r \): answer \( NO \)
- For all positions \( i \) where \( s_j \) differs from \( s \)
  - Solve subproblem with
    - \( s \) changed at position \( i \) to value \( s_j (i) \)
    - \( r = r - 1 \)
- Note: we find a solution, if and only one of these subproblems has a solution
Example

- Strings 01113, 02223, 01221, \( d=2 \) gives

  - (02113, 1)
    - (02213, 0) ☹️
    - (02123, 0) ☹️
  
  - (01213, 1)
    - (02213, 0) ☹️
    - (01223, 0) 😊

  - (01123, 1)
    - (02123, 0) ☹️
    - (01223, 0) 😊
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 $(2d)^{d+1}$
- Each step can be done in polynomial time: $O(kdL)$
- Total time is $O((2d)^{d+1} \cdot kdL)$
- Speed up possible by more clever branching and by kernelisation
More clever branching step

- Choose an \( s_j \) with Hamming distance > \( d \) to \( s \)
  - If none exists, \( s \) is a solution
- If Hamming distance of \( s_j \) to \( s \) > \( d+r \): answer \( NO \)
- Choose pick \( d+1 \) positions where \( s_j \) differs from \( s \)
  - Solve subproblem with
    - \( s \) changed at position \( i \) to value \( s_j(i) \)
    - \( r = r - 1 \)
- Still correct since the solution differs with \( s \) on at least one of these positions. \( O(kL + kd(d+1)^d) \) time
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
Kernelization
Kernelisation

- **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 $p(n) + g(f(k))$
Kernelisation

• Helps to analyze preprocessing
• Much recent research
• Today: definition and some examples
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 $A$, that transforms inputs of $P$ to inputs of $P$, such that

- $(I,k) \in P$, if and only if $A(I,k) \in P$ 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 $k$
Kernels and FPT

• **Theorem.** Consider a decidable parameterized problem. Then the problem belongs to FPT, if and only if it has a kernel

  • $\leq$ Build kernel and solve the problem on kernel
  
  • $\Rightarrow$ Suppose we have an $f(k)n^c$ algorithm. Run the algorithm for $n^{c+1}$ steps. If it did not yet solve the problem, return the input as kernel: it has size at most $f(k)$. If it solved the problem, then return small YES / NO instance
Consequence

- If a problem is $W[1]$-hard, it has no kernel, unless $FPT=W[1]$
- There also exist techniques to give evidence that problems have no kernels of polynomial size
  - If problem is *compositional* and NP-hard, then it has no polynomial kernel
  - Example is e.g., LONG PATH
First kernel: Convex string recoloring

• Application from molecular biology; NP-complete

Convex String Recoloring
– Given: string $s$ in $\Sigma^*$, integer $k$
– Parameter: $k$
– Question: can we change at most $k$ characters in the string $s$, such that $s$ becomes convex, i.e., for each symbol, the positions with that symbol are consecutive.

• Example of convex string: aaaccccbxxxxffflf
• Example of string that is not convex: abba
• Instead of symbols, we talk about colors
Kernel for convex string recoloring

- **Theorem**: Convex string recoloring has a kernel with $O(k^2)$ characters.
Notions

• Notion: good and bad colors
• A color is *good*, if it is consecutive in $s$, otherwise it is bad
• abba: a is bad and b is good
• Notion of *block*: consecutive occurrences of the same color, ie. aabbacc has four blocks
• Convex: each color has one block
Construction of kernel

• Apply three reduction rules
  – Rule 1: limit #blocks of bad colors
  – Rule 2: limit #different good colors
  – Rule 3: limit #characters in $s$ per block

• Count
Rule 1

• If there are more than $4k$ blocks of bad colors, say NO
  – Formally, transform to trivial NO-instance, e.g. (aba, 0)

  – Why correct?
    • Each change can decrease the number of blocks of bad colors by at most 4.
      – Example: abab -> abbb
Rule 2

• If we have two consecutive blocks of good colors, then change the color of the second block to that of the first

• E.g: abbbbccea -> abbbbbbbba

• Why correct?
  – We will only recolor the c’s to connect bad colors, and this still comes at the same cost.
Rule 3

• If a block has more than \( k+1 \) characters, delete all but \( k+1 \) of the block

• Correctness: a block of such a size can never be changed
Counting

• After the rules have been applied, we have at most:
  – \(4k\) blocks of bad colors
  – \(4k+1\) blocks of good colors: at most one between each pair of bad colors, one in front and one in the end
  – Each block has size at most \(k+1\)
• String has size at most \((8k+1)(k+1)\)
• This can be improved by better analysis, more rules, ...
Fixed Parameter Complexity

(2nd half)

• Reminder

• Kernelization:
  – Point line cover (warm-up)
  – Vertex Cover
  – Max Sat
  – Cluster editing
  – Non-blocker (might skip it)

• FPT techniques:
  – Iterative Compression
  – Color Coding
Reminder: Fixed Parameter Complexity

• **Given**: Graph G, integer $k$, ...

• **Parameter**: $k$

• **Question**: Does G have a ??? of size at least (at most) $k$?

• **Distinguish between behavior**:
  - $O(f(k) \times n^c)$
  - $\Omega(n^{f(k)})$
Reminder: Kernelisation

- Let $P$ be a parameterized problem. (Each input of the form $(I,k)$.)

A *reduction to a problem kernel* is an algorithm $A$, that transforms inputs of $P$ to inputs of $P$, such that

- $(I,k) \in P$, if and only if $A(I,k) \in P$ 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 $k$
Kernel: Point-line cover

**In:** $n$ points in the plane, integer $k$

**Question:** Can you hit all the points with $k$ straight lines?

NP-Complete.
Quadratic kernel

Rule 1: If some line covers \( k+1 \) points use it (and reduce \( k \) by one).

Rule 2: If Rule 1 is not applicable and \( n > k^2 \), say NO.

Kernel with \( k^2 \) points!

Kratsch Philip Ray’13: No kernel with \( O(k^{2-\varepsilon}) \) points unless \( \text{coNP} \subseteq \text{NP/poly} \)
Vertex cover: observations that help for kernelisation

- Isolated vertices can be ignored.
- If $v$ has degree at least $k+1$, $v$ belongs to each vertex cover in $G$ of size at most $k$.
  - If $v$ is not in the vertex cover, then all its neighbors are in the vertex cover.
- If all vertices have degree at most $k$, and $m > k^2$, there is no vertex cover of size $k$. 
Kernelisation for Vertex Cover

\[ H = G; \( S = \emptyset; \) \]

While there is a vertex \( v \) in \( H \) of degree at least \( k+1 \) do

- Remove \( v \) and its incident edges from \( H \)
- \( k = k - 1; \( S = S + v ; \) \)

If \( k < 0 \) then return \textbf{false}

If \( H \) has at least \( k^2+1 \) edges, then return \textbf{false}

Remove vertices of degree 0
Time

- Kernelisation step can be done in $O(n+m)$ time
- After kernelisation, we must solve the problem on a graph with at most $k^2$ edges, e.g., with branching this gives:
  - $O(n + m + 2^k k^2)$ time
  - $O(kn + 2^k k^2)$ time can be obtained by noting that there is no solution when $m > kn$. 
Better kernel for vertex cover

- Nemhauser-Trotter: kernel of at most $2k$ vertices
- Make ILP formulation of Vertex Cover
- Solve relaxation
- All vertices $v$ with $x_v > \frac{1}{2}$: put $v$ in set
- All vertices $v$ with $x_v < \frac{1}{2}$: $v$ is not in the set
- Remove all vertices except those with value $\frac{1}{2}$, and decrease $k$ accordingly
- Gives kernel with at most $2k$ vertices, but why is it correct?
Nemhauser Trotter proof plan

1. Write down the ILP for Vertex Cover
2. Write down the LP relaxation, let $x$ be an optimal solution
   - Let $X_{< \frac{1}{2}} = \{ v \in V : x_v < \frac{1}{2} \}$, $X_{> \frac{1}{2}} = \{ v \in V : x_v > \frac{1}{2} \}$
3. Given any minimum vertex cover $S$, show that $S^* := (S \cup X_{> \frac{1}{2}}) \setminus X_{< \frac{1}{2}}$ also is an optimal VC.

Correctness of kernel follows from this observation
1. ILP

\[
\min \sum_{v \in V} x_v \\
\forall \{v, w\} \in E : x_v + x_w \geq 1 \\
x_v \in \{0, 1\}
\]
2. Relaxation

\[
\min \sum_{v \in V} x_v
\]

\[
\forall \{v, w\} \in E : x_v + x_w \geq 1
\]

\[
x_v \geq 0
\]
3. Observation

- If $S$ is min. $vc$, so is $S^* := (S \cup X_{\geq \frac{1}{2}}) \setminus X_{<\frac{1}{2}}$.

$$\forall \{v, w\} \in E : x_v + x_w \geq 1$$
3. Observation

- If $S$ is min. $\text{vc}$, so is $S^* := (S \cup X_{\geq \frac{1}{2}}) \setminus X_{< \frac{1}{2}}$.

- $S^*$ is a $\text{vc}$ since $N(X_{< \frac{1}{2}})$ is in $S^*$.

- Optimize $|X_{\geq \frac{1}{2}} \setminus S| \cap |S|$. Constraints: 
  - Add $\varepsilon$ to $x_v$ if $v \in X_{\geq \frac{1}{2}} \cap S$; subtract $\varepsilon$ if $v \in S \setminus X_{\geq \frac{1}{2}}$. 

---

Fixed Parameter Complexity
3. Observation

- If $S$ is min. vc, so is $S^* := (S \cup X_{\geq \frac{1}{2}}) \setminus X_{\leq \frac{1}{2}}$.

- $S^*$ is a vc since $N(X_{\leq \frac{1}{2}})$ is in $S^*$

- Assume $|X_{\geq \frac{1}{2}} \setminus S| > |X_{\leq \frac{1}{2}} \cap S|$. Contradiction!

  - $x$ is not optimal! Let $\varepsilon = \min_{v \in X_{\geq \frac{1}{2}}} x_v - \frac{1}{2}$;

  - Add $\varepsilon$ to $x_v$ if $v \in X_{\leq \frac{1}{2}} \cap S$; subtract $\varepsilon$ if $v \in S \setminus X_{\geq \frac{1}{2}}$.  


2k kernel for Vertex Cover

- Solve the relaxation (polynomial time with the ellipsoid method, practical with Simplex)
- If the relaxation has optimum more than k, then say no
- Otherwise, get rid of the 0’s and 1’s, decrease k accordingly
- At most 2k vertices have weight ½ in the relaxation
- So, kernel has 2k vertices.
- It can (and will often) have a quadratic number of edges
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 with $C$
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
    - Reason: if $a+b \geq C$, either $a \geq c/2$ or $b \geq c/2$
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 $L \geq k$: answer is **YES**
  - Select in each of the first $k$ long clauses a literal, whose complement is not yet selected
  - Set these all to true
  - At least $k$ long clauses are satisfied
Reducing to only short clauses

- If less than $k$ long clauses
  - Make new instance, with only the short clauses and the parameter set to $k-L$
  - Valid, because: 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
    - $\Rightarrow$: 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
    - $\Leftarrow$: there are only $L$ long clause so we satisfy $k-L$ short ones
An $O(k^2)$ kernel for Maximum Satisfiability

- **If** at least $2k$ clauses **then** return YES
- **If** at least $k$ long clauses **then** return YES
- **Else**
  - remove all L long clauses
  - set $k = k - L$
Reminder: Cluster editing

- **Instance**: undirected graph $G=(V,E)$, integer $K$
- **Parameter**: $K$
- **Question**: can we make at most $K$ modifications to $G$, such that each connected component is a clique, where each modification is an addition of an edge or the deletion of an edge?
Kernelisation for cluster editing

- Quadratic kernel
- General form:
- Repeat rules, until no rule is possible
  - Rules can do some necessary modification and decrease $k$ by one
  - Rules can remove some part of the graph
  - Rules can output YES or NO
Trivial rules and plan

- **Rule 1**: If a connected component of $G$ is a clique, remove this connected component.
- **Rule 2**: If we have more than $k$ connected components and Rule 1 does not apply: Answer NO.
- **Consequence**: after Rule 1 and Rule 2, there are at most $k$ connected components.
- **Plan**: find rules that make connected component small.
- We change the input: some pairs are permanent and others are forbidden.
Observation and rule 3

- If two vertices \( v, w \) have \( k+1 \) neighbors in common, they must belong to the same clique in a solution

**Rule 3:** If \( v, w \) have \( k+1 \) neighbors in common, then
  - If \( \{v,w\} \) did not exist, add it, and set \( k = k - 1 \).
  - Set the edge \( \{v,w\} \) to be **permanent**
Another observation and rule 4

• If there are at least \( k+1 \) vertices that are adjacent to exactly one of \( v \) and \( w \), then \( \{v,w\} \) cannot be an edge in the solution

Rule 4: If there are at least \( k+1 \) vertices that are adjacent to exactly one of \( v \) and \( w \), then

– If \( \{v,w\} \) is an edge: delete it and decrease \( k \) by one

– Mark the pair \( \{v,w\} \) as forbidden
A trivial rule

- Rule 5: if a pair is forbidden and permanent then there is no solution
Transitivity

- **Rule 6**: if \( \{v, w\} \) is permanent, and \( \{w, x\} \) is permanent, then set \( \{v, x\} \) to be permanent (if the edge was nonexisting, add it, and decrease \( k \) by one)

- **Rule 7**: if \( \{v, w\} \) is permanent and \( \{w, x\} \) is forbidden, then set \( \{v, x\} \) to be forbidden (if the edge existed, delete it, and decrease \( k \) by one)
Counting

• Rules can be executed in polynomial time
• One can find in $O(n^3)$ time an instance to which no rules apply (with properly chosen data structures)

Claim: If no rule applies, and the instance is a YES-instance, then every connected component has size at most $4k$.

Proof:
• Suppose a YES-instance, and consider a connected component $C$ of size at least $4k+1$.
• At least $2k+1$ vertices are not involved in a modification, say this is the set $W$
• $W$ must form a clique, and all edges in $W$ become permanent (Rule 3) …
Counting continued

- Each vertex in C-W that is incident to \( k+1 \) or more vertices in \( W \) has a permanent edge to a vertex in \( W \) (r3), and then gets permanent edges to all vertices in \( W(r6) \), and then becomes member of \( W \)
- Each vertex in C-W for which at least \( k+1 \) vertices in \( W \) are not adjacent: it gets a forbidden edge to each vertex in \( W \) (r4,r7)
- Each vertex in C-W is handled as \( |W|>2k; \ C-W=\emptyset \)
  - A vertex in C-W has either \( k+1 \) existing or missing edges
- So, the vertices in C-W will not belong to the connected component, and the vertices in \( W \) form a clique, which then is removed by Rule 1
Conclusion

• Rule 8: If no other rule applies, and there is a connected component with at least $4k+1$, vertices, say NO.

• As we have at most $k$ connected components, each of size at most $4k$, our kernel has at most $4k^2$. 
Comments

• This argument is due to Gramm et al.
• Better and more recent algorithms exist: faster algorithm \((2.7^k)\) and linear kernels
Non-blocker

- Given: graph $G=(V,E)$, integer $k$
- Parameter $k$
- Question: Does $G$ have a dominating set of size at most $|V|-k$?
Lemma and simple linear kernel

- If $G$ does not have vertices of degree 0, then $G$ has a dominating set with at most $|V|/2$ vertices
  - Proof: per connected component: build spanning tree. The vertices on the odd levels form a ds, and the vertices on the even levels form a ds. Take the smallest of these.

- $2k$ kernel for non-blocker after removing vertices of degree 0
Improvements

• Lemma (Blank and McCuaig, 1973) If a connected graph has minimum degree at least two and at least 8 vertices, then the size of a minimum dominating set is at most $2|V|/5$.

• Lemma (Reed) If a connected graph has minimum degree at least three, then the size of a minimum dominating set is at most $3|V|/8$.

• Can be used by applying reduction rules for killing degree one and degree two vertices
Iterative compression
Feedback Vertex Set

- **Instance**: graph $G=(V,E)$
- **Parameter**: integer $k$
- **Question**: Is there a set of at most $k$ vertices $W$, such that $G-W$ is a forest?
  - Known in FPT
  - Here: recent algorithm $O(5^k p(n))$ time algorithm
  - Can be done in $O(5^k kn)$ or less with kernelisation
  - Fastest known algorithm (randomized): $O(3^k p(n))$, invented at Utrecht, using treewidth techniques.
Iterative compression technique

- Number vertices $v_1, v_2, \ldots, v_n$
- Let $X = \{v_1, v_2, \ldots, v_k\}$
- for $i = k+1$ to $n$ do
  - Add $v_i$ to $X$
    - Note: $X$ is a FVS of size at most $k+1$ of $G[v_1, v_2, \ldots, v_i]$]
  - Call a subroutine that either
    - Finds (with help of X) a feedback vertex set $Y$ of size at most $k$ in $\{v_1, v_2, \ldots, v_i\}$; set $X = Y$ OR
    - Determines that such $Y$ does not exist; stop, return NO
Compression subroutine

- **Given**: graph $G$, FVS $X$ of size $k + 1$
- **Question**: find if existing FVS of size $k$
  - Is subroutine of main algorithm

- **for all subsets $S$ of $X$ do**
  - Determine if there is a FVS of size at most $k$ that contains all vertices in $S$ and no vertex in $X - S$
Yet a deeper subroutine

- Given: Graph G, FVS X of size $k+1$, set S
- Question: find if existing a FVS of size $k$ containing all vertices in S and no vertex from $X - S$

1. Remove all vertices in S from G
2. Mark all vertices in $X - S$
3. If marked cycles contain a cycle, then return NO
4. While marked vertices are adjacent, contract them
5. Set $k = k - |S|$. If $k < 0$, then return NO
6. If G is a forest, then return YES; S
7. ...

Universiteit Utrecht
Answer NO, we cannot kill this cycle since we don’t remove any vertex.
Compression subroutine

$G\{v_1, v_2, ..., v_i\}$
Subroutine continued

7. If an unmarked vertex $v$ has at least two edges to marked vertices
   - If these edges are parallel, i.e., to the same neighbor, then $v$ must be in a FVS (we have a cycle with $v$ the only unmarked vertex)
   - Put $v$ in $S$, set $k = k - 1$ and recurse
   - Else recurse twice:
     - Put $v$ in $S$, set $k = k - 1$ and recurse
     - Mark $v$, contract $v$ with all marked neighbors and recurse
       - The number of marked vertices is one smaller
Compression subroutine

\[ G[\{v_1, v_2, \ldots, v_i\}] \]
Compression subroutine

\[ G[\{v_1, v_2, \ldots, v_i\}] \]

Fixed Parameter Complexity
Compression subroutine

\[ G[\{v_1, v_2, \ldots, v_i\}] \]
Other case

All unmarked vertices have at most 2 marked neighbors!

8. Choose an unmarked vertex \( v \) that has at most one unmarked neighbor (a leaf in \( G[V-X] \))
   - By step 7, it also has at most one marked neighbor
   - If \( v \) is a leaf in \( G \), then remove \( v \)
   - If \( v \) has degree 2, then remove \( v \) and connect its neighbors
Other case

$G[\{v_1, v_2, ..., v_i\}]$
Analysis

• Precise analysis gives $O^*(5^k)$ subproblems in total
• Imprecise: $2^k$ subsets $S$
• Only branching step:
  – $k$ is decreased by one, or
  – Number of marked vertices is decreased by one
• Initially: number of marked vertices + $k$ is at most $2k$
• Bounded by $2^k \cdot 2^{2k} = 8^k$
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?
Long Path on Directed Acyclic Graphs (DAG)

• Consider long path on DAG's.

• Polynomial time easily!
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 $O(k! \ (n+m))$ (try all orders of colors and reduce to long paths in DAG)
  - Better: $O(2^k \ (nm))$ time using dynamic programming

- **Used as subroutine…**
DP

- **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
  - If there is a path of length \( k \), we find one with positive probability (\( 2^k / k! \approx e^{-k} \))
  - We can do this check in \( O(2^k nm) \) time and repeat it \( 10 e^k \) times to get probability \( 1 - (1 - e^{-k}) e^k \geq 1 - \frac{1}{e} \) to find the path (using \( (1 - x^{-1})^x \leq 1/e \)).
  - Gives \( O((2e)^k(n+m) \text{ time}) \) running time.
Conclusions

• Similar techniques work (usually much more complicated) for many other problems
• $W[\ldots]$-hardness results indicate that FPT-algorithms do not exist for other problems
• Note similarities and differences with exponential time algorithms