What's next? Reductions other than kernelization

Dániel Marx

Humboldt-Universität zu Berlin

(with help from Fedor Fomin,

Daniel Lokshtanov and Saket Saurabh)

WorKer 2010: Workshop on Kernelization

Nov 9, 2010

Kernelization

The story:

- We want to obtain FPT results.
- Kernelization is a nice and useful technique for obtaining FPT results.
- Kernelization + brute force (exact algorithm) can show that a problem is FPT.
- Which problems can be solved by kernelization + brute force?

Kernelization

Lemma: Every FPT problem has a kernel.

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.

□

Kernelization

Lemma: Every FPT problem has a kernel.

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.

□

Every FPT result can be obtained as kernelization + brute force!

FPT	=	Kernelization	+	Exact algorithms

Efficiency

k-PATH: Given graph G and integer k, does G has a path of length k?

- $O^*(2^{O(k \log k)})$ algorithm using representative systems [Monien 1985].
- $O^*((2e)^k)$ randomized algorithm by color coding [Alon-Yuster-Zwick 1995]
- Color coding can be derandomized to $O^*(2^{O(k)})$.
- $O^*(2^k)$ randomized algebraic algorithm [Williams 2009]
- $O^*(1.66^k)$ randomized algebraic algorithm [Björklund 2010]

Efficiency

k-PATH: Given graph G and integer k, does G has a path of length k?

- $O^*(2^{O(k \log k)})$ algorithm using representative systems [Monien 1985].
- $O^*((2e)^k)$ randomized algorithm by color coding [Alon-Yuster-Zwick 1995]
- Color coding can be derandomized to $O^*(2^{O(k)})$.
- $O^*(2^k)$ randomized algebraic algorithm [Williams 2009]
- $O^*(1.66^k)$ randomized algebraic algorithm [Björklund 2010]

Question of efficiency:

Can we obtain a $O^*(2^{O(k)})$ or a $O^*(2^{\text{poly}(k)})$ time algorithm for k-PATH by a combination of kernelization and exact algorithms?

ETH

Exponential Time Hypothesis (ETH):
n-variable 3SAT cannot be solved in time $2^{o(n)}$.

Formulated by Impagliazzo, Paturi, and Zane in 2001, since then many lower bounds were proved based on ETH (equivalent to FPT $\neq$ M[1] and implies FPT $\neq$ W[1]).

ETH

Exponential Time Hypothesis (ETH):
n-variable 3SAT cannot be solved in time $2^{o(n)}$.

Formulated by Impagliazzo, Paturi, and Zane in 2001, since then many lower bounds were proved based on ETH (equivalent to FPT $\neq$ M[1] and implies FPT $\neq$ W[1]).

Lower bound for k-PATH:

- **Sparsification Lemma:** Assuming ETH, m-clause 3SAT cannot be solved in time $2^{o(m)}$.
- The k-PATH problem is NP-hard: an m-clause 3SAT formula can be reduced to a k-PATH instance with $k = O(m)$ and $O(m)$ edges/vertice.
- Assuming ETH, there is no $O^*(2^{o(k)})$ time parameterized algorithm and no $O^*(2^{o(|E(G)|)})$ or $O^*(2^{o(|V(G)|)})$ time exact algorithm for k-PATH.

Efficiency

We have seen: Assuming ETH, there is no $2^{o(|V(G)|)}$ exact algorithm for k-PATH.

Thus if we want a $O^*(2^{O(k^c)})$ time algorithm by kernelization + exact algorithm, then we need a kernel with $|V(G)| = O(k^c)$.

Efficiency

We have seen: Assuming ETH, there is no $2^{o(|V(G)|)}$ exact algorithm for k-PATH.

Thus if we want a $O^*(2^{O(k^c)})$ time algorithm by kernelization + exact algorithm, then we need a kernel with $|V(G)| = O(k^c)$.

Theorem: [Bodlaender et al. 2008] k-PATH has no poly kernel (unless the polynomial hierarchy collapses).

Kernelization + exact algorithm cannot give a $O^*(2^{O(k^c)})$ time algorithm for k-PATH!

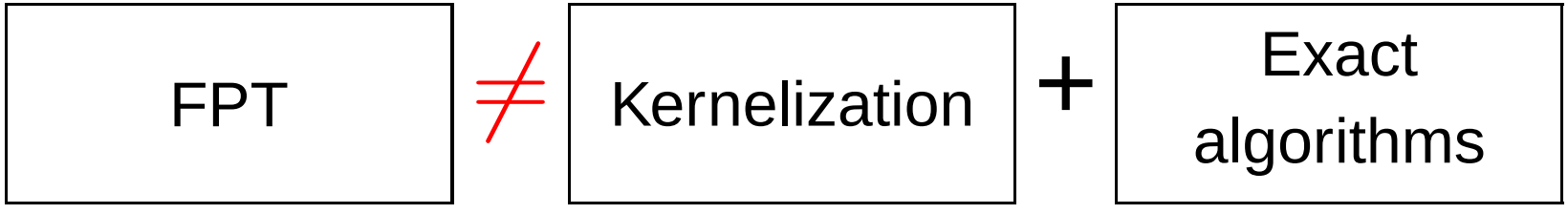
Efficiency

We have seen: Assuming ETH, there is no $2^{o(|V(G)|)}$ exact algorithm for k-PATH.

Thus if we want a $O^*(2^{O(k^c)})$ time algorithm by kernelization + exact algorithm, then we need a kernel with $|V(G)| = O(k^c)$.

Theorem: [Bodlaender et al. 2008] k-PATH has no poly kernel (unless the polynomial hierarchy collapses).

Kernelization + exact algorithm cannot give a $O^*(2^{O(k^c)})$ time algorithm for k-PATH!



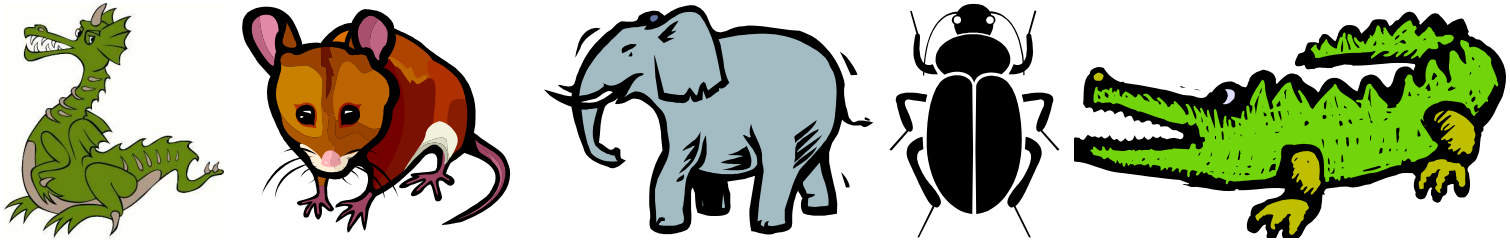
Kernelization

The extended story:

- We want to obtain FPT results.
- Kernelization is a nice and useful technique for obtaining FPT results.
- Kernelization + brute force (exact algorithm) can show that a problem is FPT.
- Every FPT membership result can be obtained this way, but for some problems this approach does not give the most efficient FPT algorithms.
- We have lower bound techniques to show that kernelization is not the right approach for some problems (or at least it does not tell the full story).

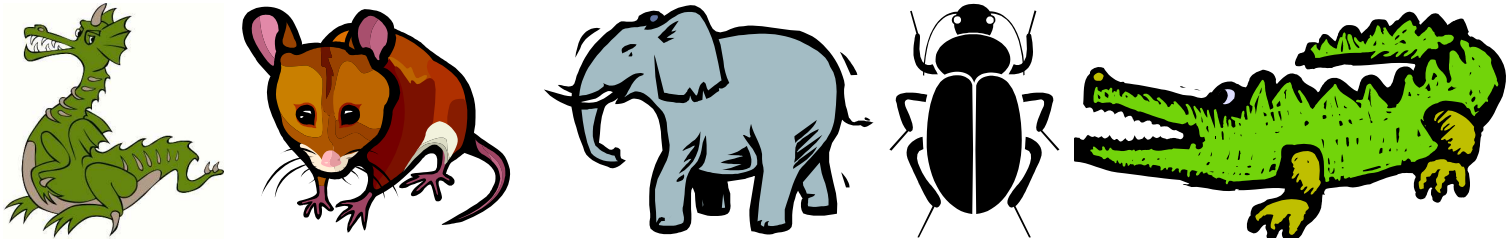
Tools vs. problems

We have a list of problems that we want to solve:



Tools vs. problems

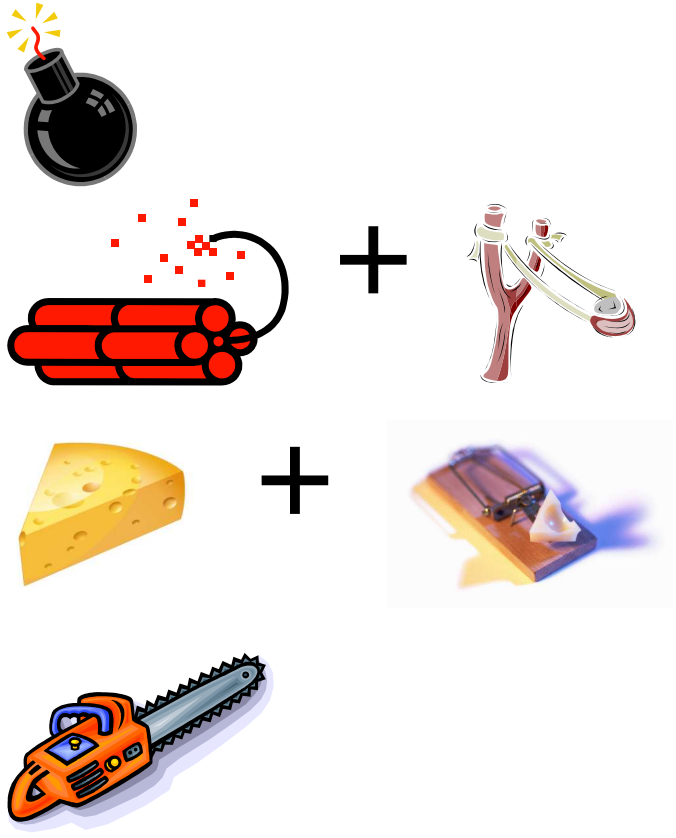
We have a list of problems that we want to solve:



And we have a set of techniques:

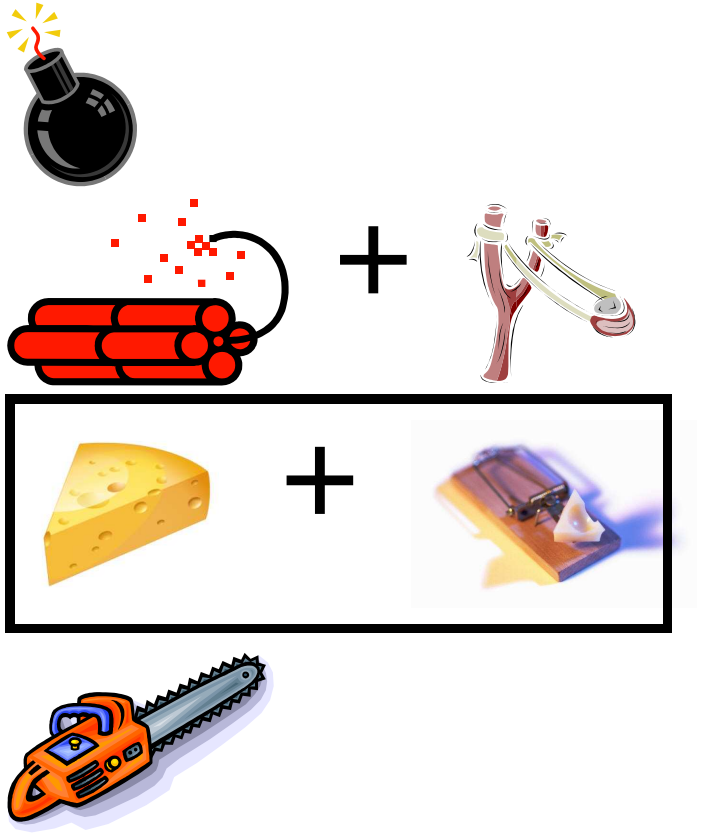
Tools vs. problems

A problem can be solved by several (combination of) techniques...



Tools vs. problems

A problem can be solved by several (combination of) techniques...



...but not all of them are the right way of solving the problem.

Efficiency

An example where kernelization + brute force is efficient:

- k-VERTEX COVER has a $2k$-vertex kernel and can be solved in $O^*(2^{|V(G)|})$ time by brute force
- $O^*(4^k)$ time algorithm.

Efficiency

An example where kernelization + brute force is efficient:

- k-VERTEX COVER has a $2k$-vertex kernel and can be solved in $O^*(2^{|V(G)|})$ time by brute force
- $O^*(4^k)$ time algorithm.

An example where kernelization + exact algorithms is currently not efficient:

- Current best kernel for FEEDBACK VERTEX SET has $O(k^2)$ edges/vertices [Thomassé 2009]
- Assuming ETH, there is no $2^{o(n)}$ exact algorithm.
- Currently $O^*(c^{k^2})$ is the best possible by kernelization + exact algorithm, but can be solved in time $O^*(c^k)$ by other techniques.

Branching

Bounded depth search trees is one of the most basic FPT techniques.

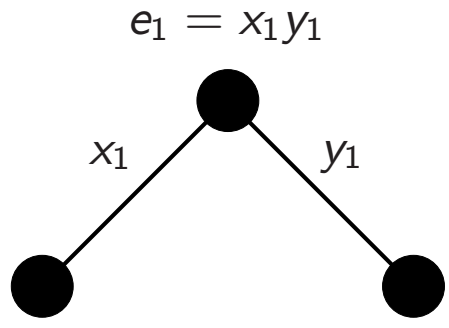
Example: $O^*(2^k)$ algorithm for vertex cover.

$e_1 = x_1 y_1$

Branching

Bounded depth search trees is one of the most basic FPT techniques.

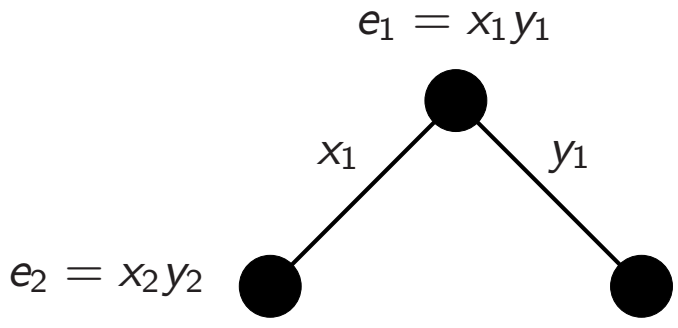
Example: $O^*(2^k)$ algorithm for vertex cover.



Branching

Bounded depth search trees is one of the most basic FPT techniques.

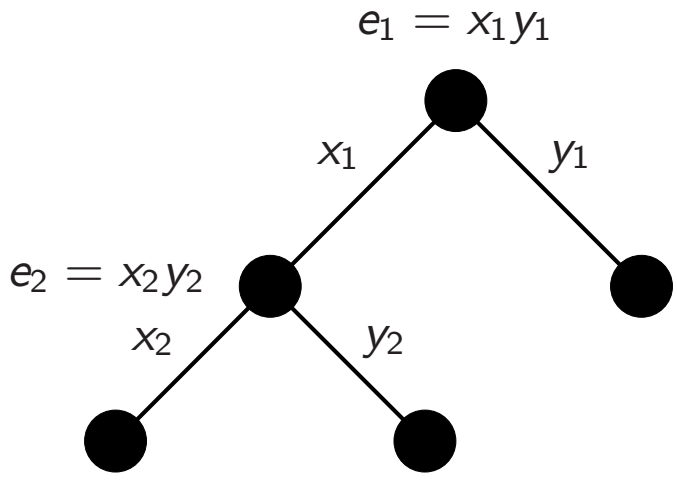
Example: $O^*(2^k)$ algorithm for vertex cover.



Branching

Bounded depth search trees is one of the most basic FPT techniques.

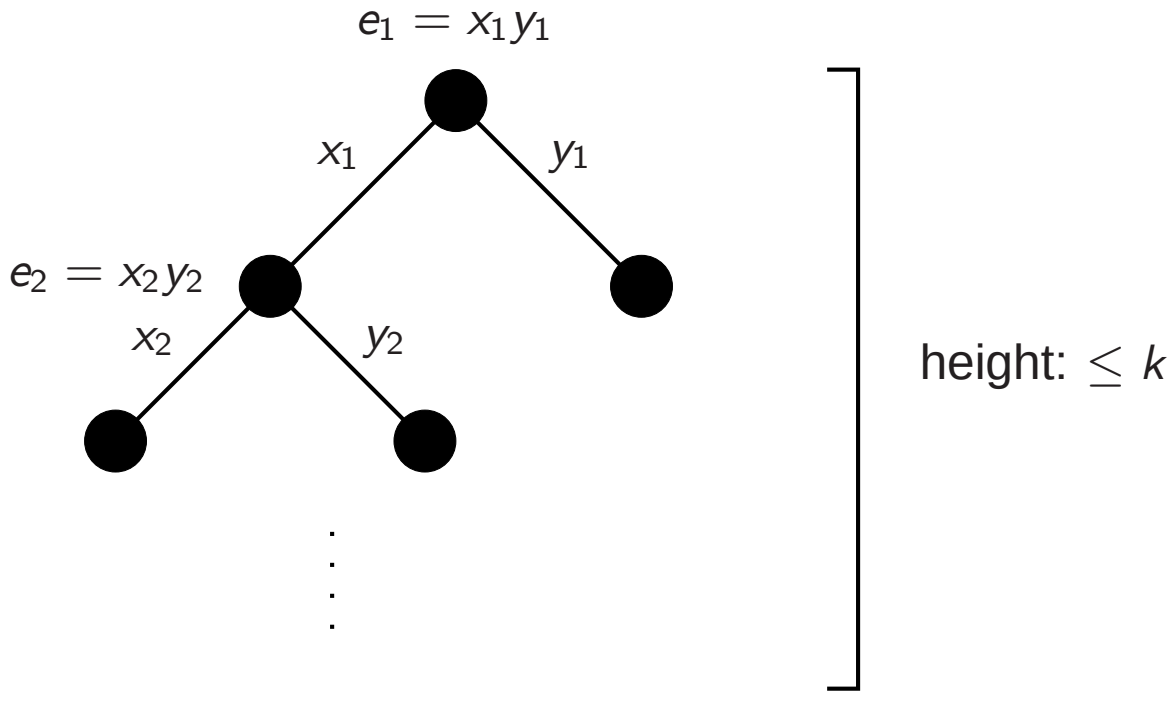
Example: $O^*(2^k)$ algorithm for vertex cover.



Branching

Bounded depth search trees is one of the most basic FPT techniques.

Example: $O^*(2^k)$ algorithm for 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.

Branching

A typical technique: a NECESSARY SET is a set N of elements such that every solution contains at least one element of N

$\Rightarrow$ branch on the elements on N.

- $|N|$ is constant $\Rightarrow O^*(c^k)$ algorithm.
- $|N| = O(k) \Rightarrow O^*(k^k) = O^*(2^{O(k \log k)})$ algorithm.
- $|N| = O(\log n) \Rightarrow O^*(\log^k n)$ algorithm

Branching

A typical technique: a NECESSARY SET is a set N of elements such that every solution contains at least one element of N

$\Rightarrow$ branch on the elements on N.

- $|N|$ is constant $\Rightarrow O^*(c^k)$ algorithm.
- $|N| = O(k) \Rightarrow O^*(k^k) = O^*(2^{O(k \log k)})$ algorithm.
- $|N| = O(\log n) \Rightarrow O^*(\log^k n)$ algorithm

We will focus on $O^*(c^k)$ type branching algorithms!

Which problems can be solved (efficiently) by branching?

Branching rules

Definition: A **branching rule** for a parameterized problem P is a polynomial-time algorithm that given an instance (I, k) with $k > 1$, produces instances $(I_1, k_1), \ldots, (I_c, k_c)$ such that

- $|I_j| \leq |I|$ for every $1 \leq j \leq c$,
- $k_j < k$ for every $1 \leq j \leq c$, and
- (I, k) is a yes-instance if and only if there is a $1 \leq j \leq c$ such that (I_j, k_j) is a yes-instance,

for some constant c.

Observation: Given a branching rule for P, we can solve the problem in time $O^*(c^k)$ by a bounded search tree algorithm.

Example

WEIRD VERTEX COVER

Input: A graph G and an integer k

Output: "YES" if G has a vertex cover of size k and k is a power of 2

"NO" otherwise

Does WEIRD VERTEX COVER have a branching rule?

Example

WEIRD VERTEX COVER

Input: A graph G and an integer k

Output: "YES" if G has a vertex cover of size k and k is a power of 2

"NO" otherwise

Does WEIRD VERTEX COVER have a branching rule?

Observation: If the branching rule produces $(I_1, k_1), \ldots, (I_c, k_c)$ from (I, k), then $k_j \leq k/2$ for every j (otherwise k_j is not a power of 2).

$\Rightarrow$ The height of the search tree is $\leq \log k$.

$\Rightarrow$ The size of the search tree is polynomial in k.

$\Rightarrow$ No branching rule unless P $=$ NP.

Example

WEIRD VERTEX COVER	
Input:	A graph G and an integer k
Output:	"YES" if G has a vertex cover of size k and k is a power of 2
	"NO" otherwise

Does WEIRD VERTEX COVER have a branching rule?

Observation: If the branching rule produces $(I_1, k_1), \ldots, (I_c, k_c)$ from (I, k), then $k_j \leq k/2$ for every j (otherwise k_j is not a power of 2).

$\Rightarrow$ The height of the search tree is $\leq \log k$.

$\Rightarrow$ The size of the search tree is polynomial in k.

$\Rightarrow$ No branching rule unless P $=$ NP.

This is very wrong: this problem should be solvable by branching.

Linear polynomial parameter transformation

Definition: A **linear polynomial parameter transformation (LPPT)** from problem P_1 to P_2 is a mapping that maps instance (I_1, k_1) to instance (I_2, k_2) such that

- $(I_1, k_1) \in P_1$ if and only if $(I_2, k_2) \in P_2$,
- $k_2 = O(k_1)$, and
- the transformation can be computed in polynomial time.

Definition: The class BranchFPT contains a parameterized problem P if there is a linear polynomial parameter transformation from P to a problem that has a branching rule.

WEIRD VERTEX COVER is LPPT-reducible to VERTEX COVER $\in$ BranchFPT.

$\Rightarrow$ WEIRD VERTEX COVER is in BranchFPT.

k-PATH

k-PATH: Is there a path of length k in G.

The known $O^*(2^{O(k)})$ time algorithms (color coding, divide and color, algebraic techniques) are not branching algorithms!

Open Question #1:

Is k-PATH in BranchFPT?

k-PATH

k-PATH: Is there a path of length k in G.

The known $O^*(2^{O(k)})$ time algorithms (color coding, divide and color, algebraic techniques) are not branching algorithms!

Open Question #1:

Is k-PATH in BranchFPT?

Open Question #2: [Paturi]

Is there a polynomial time algorithm for k-PATH with $2^{-O(k)}$ success probability?

k-PATH

k-PATH: Is there a path of length k in G.

The known $O^*(2^{O(k)})$ time algorithms (color coding, divide and color, algebraic techniques) are not branching algorithms!

Open Question #1:	Open Question #2: [Paturi]
Is k-PATH in BranchFPT?	Is there a polynomial time algorithm for k-PATH with $2^{-O(k)}$ success probability?

Observation: A positive answer for the first question implies a positive answer for the second!

(A branching algorithm can be turned into a randomized algorithm.)

Relation to linear-vertex kernels

Observation: If problem P has a linear vertex-kernel and P parameterized by the number of vertices can be solved by branching, then P is in BranchFPT: there is an LPPT-reduction to a problem that can be solved by branching.

Relation to linear-vertex kernels

Observation: If problem P has a linear vertex-kernel and P parameterized by the number of vertices can be solved by branching, then P is in BranchFPT: there is an LPPT-reduction to a problem that can be solved by branching.

Example:
MAX INTERNAL SPANNING TREE: Does G have a spanning tree with at least k internal vertices?

A $3k$-vertex kernel is known [Fomin-Gaspers-Saurabh-Thomassé 2009].

It is not obvious if MAX INTERNAL SPANNING TREE parameterized by the number of vertices is in BranchFPT!

Open Question #3: Is MAX INTERNAL SPANNING TREE in BranchFPT?

Nondeterministic bits

Observation: A parameterized problem P is in BranchFPT if and only if it has an NP characterization with $O(k)$ size certificates.

Nondeterministic bits

Observation: A parameterized problem P is in BranchFPT if and only if it has an NP characterization with $O(k)$ size certificates.

Proof:

$\Rightarrow$: Let the certificate be the decisions we make during branching. Can be encoded in $O(k)$ bits and can be verified in polynomial time.

Nondeterministic bits

Observation: A parameterized problem P is in BranchFPT if and only if it has an NP characterization with $O(k)$ size certificates.

Proof:

$\Rightarrow$: Let the certificate be the decisions we make during branching. Can be encoded in $O(k)$ bits and can be verified in polynomial time.

$\Leftarrow$: Let us define the variant P', where $(x, k, w, \ell) \in P' \iff w$ can be extended with at most ℓ more bits to a witness of (x, k).

- There is an LPPT-reduction from P parameterized by k to P' parameterized by ℓ: $(x, k) \in P \iff (x, k, \emptyset, O(k)) \in P'$
- P' parameterized by ℓ can be solved by branching: guess the next bit of the witness and decrease ℓ.

Feedback Vertex Set

FEEDBACK VERTEX SET: Is there a set S of at most k vertices such that $G \setminus S$ contains no cycle?

- Randomized $O^*(4^k)$ branching algorithm by [Becker, Bar-Yehuda, Geiger 2000].
- First deterministic $O^*(2^{O(k)})$ time algorithm in by using iterative compression [Dehne et al. 2005] [Guo et al. 2005].

Feedback Vertex Set

FEEDBACK VERTEX SET: Is there a set S of at most k vertices such that $G \setminus S$ contains no cycle?

- Randomized $O^*(4^k)$ branching algorithm by [Becker, Bar-Yehuda, Geiger 2000].
- First deterministic $O^*(2^{O(k)})$ time algorithm in by using iterative compression [Dehne et al. 2005] [Guo et al. 2005].

FEEDBACK VERTEX SET COMPRESSION: Given G and set of vertices S_0 such that $G \setminus S_0$ contains no cycle, is there a set S of at most k vertices such that $G \setminus S$ contains no cycle?

Lemma: FEEDBACK VERTEX SET COMPRESSION is FPT param. by $|S_0|$.

Two ways of using this: (1) iterative compression or (2) polynomial time 2-approximation.

Feedback Vertex Set

FEEDBACK VERTEX SET COMPRESSION: Given G and set of vertices S_0 such that $G \setminus S_0$ contains no cycle, is there a set S of at most k vertices such that $G \setminus S$ contains no cycle?

Lemma: FEEDBACK VERTEX SET COMPRESSION is FPT param. by $|S_0|$.

- Inspection of proof shows that the compression problem is in BranchFPT.
- The 2-approximation gives an LPPT from FEEDBACK VERTEX SET to the compression problem.
- $\Rightarrow$ FEEDBACK VERTEX SET is in BranchFPT.

More generally: If a constant-factor approximation can be obtained in polynomial time and the compression problem can be solved by branching, then the problem is in BranchFPT.

Relation to counting

We can define a stronger version of branching that is capable of counting the number of solutions: the number of solutions is the sum of the number of solutions in the branches.

Not all branching rules are like this.

- If there is a set N of elements such that every solution has to contain at least one element of N, then we can branch on the elements on N.
  $\Rightarrow$ Usually good for counting.
- If there is a set N of elements such that at least one solution has to contain at least one element of N, then we can branch on the elements on N.
  $\Rightarrow$ Usually bad for counting.

Relation to counting

We can define a stronger version of branching that is capable of counting the number of solutions: the number of solutions is the sum of the number of solutions in the branches.

Theorem: [Flum-Grohe 2004] #k-PATH is #W[1]-hard.

$\Rightarrow$ If k-Path can be solved by branching, then it is unlikely that the branching is good for counting.

Is it possible that hardness results for counting are the key to negative results on branching?

Further open questions

Which of the following problems is in BranchFPT?

- CONNECTED VERTEX COVER
- STEINER TREE parameterized by number of terminals
- HITTING SET parameterized by number of sets
- ...

Why not k-way branching?

We could set up definitions that allow branching into $O(k)$ directions (yielding $O^*(k^{O(k)})$ time algorithms.

Alternate definition: A **branching rule** for a parameterized problem P is a polynomial-time algorithm that given an instance (I, k) with $k > 1$, produces instances $(I_1, k_1), \ldots, (I_c, k_c)$ such that

- $c \leq k^d$,
- $|I_j| \leq |I|$ for every $1 \leq j \leq c$,
- $k_j < k$ for every $1 \leq j \leq c$, and
- (I, k) is a yes-instance if and only if there is a $1 \leq j \leq c$ such that (I_j, k_j) is a yes-instance,

for some constants c and d.

k-PATH is LPPT-reducible to a problem that admits this type of branching!

Branching for k-PATH

A polynomial-time solvable problem:

> COLORED k-PATH:
> Given a graph G with vertices colored by $[k]$ and a vector $(c_1, \ldots, c_k)$ of distinct colors, find a path of length k such that the i-th vertex has color c_i.

A generalization solvable by k-way branching:

> GENERALIZED COLORED k-PATH:
> Given a family $\mathcal{F}$ of colorings $[k^2]$ and a vector $(c_1, \ldots, c_k)$ with $c_i \in [k^2] \cup \bigstar$, find a solution by choosing a coloring from $\mathcal{F}$ and extending the vector to k distinct colors by replacing the "wildcards" $\bigstar$ by arbitrary colors in $[k^2]$.

Solvable by branching parameterized by k + number of wildcards (replace one wildcard with one of the k^2 colors).

Branching for k-PATH

GENERALIZED COLORED k-PATH:
Given a family $\mathcal{F}$ of colorings $[k^2]$ and a vector $(c_1, \ldots, c_k)$ with $c_i \in [k^2] \cup \bigstar$, find a solution by choosing a coloring from $\mathcal{F}$ and extending the vector to k distinct colors by replacing the "wildcards" $\bigstar$ by arbitrary colors in $[k^2]$.

An (n, k, k^2)-splitter is a family of functions $[n] \to [r^2]$ such that for every r-element $X \subseteq [n]$, it contains a function that is injective on X.

Theorem: [Naor, Schulman, Srinivasan 1995] There is an explicit construction of an (n, r, r^2)-splitter family containing $O(r^6 \log r \log n)$ functions.

Branching for k-PATH

GENERALIZED COLORED k-PATH:
Given a family $\mathcal{F}$ of colorings $[k^2]$ and a vector $(c_1, \ldots, c_k)$ with $c_i \in [k^2] \cup \bigstar$, find a solution by choosing a coloring from $\mathcal{F}$ and extending the vector to k distinct colors by replacing the "wildcards" $\bigstar$ by arbitrary colors in $[k^2]$.

An (n, k, k^2)-splitter is a family of functions $[n] \to [r^2]$ such that for every r-element $X \subseteq [n]$, it contains a function that is injective on X.

Theorem: [Naor, Schulman, Srinivasan 1995] There is an explicit construction of an (n, r, r^2)-splitter family containing $O(r^6 \log r \log n)$ functions.

Claim: k-PATH is LPPT-reducible to GENERALIZED COLORED k-PATH.

Proof: Let $\mathcal{F}$ be a (n, k, k^2) and let the vector be $(\bigstar, \ldots, \bigstar)$. This is a yes-instance if and only if there is a path of length k.

Treewidth reduction

Most problems are easy on bounded-treewidth graphs, with notable exceptions.

General strategy for showing that a problem is FPT:

- Solve the problem using standard techniques if treewidth is small.
- Do something if treewidth is large (answer is trivial or some reduction is possible).

A classical example

MINOR CONTAINMENT: Given graphs H and G, is H a minor of G?

Theorem: [Robertson-Seymour, Graph Minors XIII] [Kawarabayashi-Wollan 2010] MINOR CONTAINMENT is FPT parameterized by $|V(H)|$.

- If treewidth is small, then we solve the problem using standard techniques.
- Otherwise, we identify an **irrelevant vertex** whose deletion provably does not change the problem.
- Weak Structure Theorem: we can find either a large clique minor or a large "flat wall."

A classical example

MINOR CONTAINMENT: Given graphs H and G, is H a minor of G?

Theorem: [Robertson-Seymour, Graph Minors XIII] [Kawarabayashi-Wollan 2010] MINOR CONTAINMENT is FPT parameterized by $|V(H)|$.

- If treewidth is small, then we solve the problem using standard techniques.
- Otherwise, we identify an **irrelevant vertex** whose deletion provably does not change the problem.
- Weak Structure Theorem: we can find either a large clique minor or a large "flat wall."
  - Large clique minor: the problem is trivial, or a vertex of the clique minor is irrelevant.
  - Large flat wall: a "middle vertex" of the wall is irrelevant.

Irrelevant vertex

Examples of treewidth reduction using the irrelevant vertex technique:

- MINOR CONTAINMENT [Robertson-Seymour, Graph Minors XIII] [Kawarabayashi-Wollan 2010]
- CROSSING NUMBER [Grohe 2004] [Kawarabayashi-Reed 2007]
- APEX NUMBER [M.-Schlotter 2007] [Kawarabayashi 2008]
- CHORDAL DELETION [M. 2006]

Somehow these examples are too complicated for formal treatment.

Let's see some simpler examples.

LONG CYCLE

LONG CYCLE: Is there a cycle of length at least k in G?

- Treewidth reduction: In polynomial time, we can either find a cycle of length at least k or a tree decomposition of width $O(k)$.
- LONG CYCLE on a tree decomposition of width w can be solved in time $O^*(2^{O(w \log w)})$.
- Using a polynomial reduction + treewidth technique we can solve LONG CYCLE in time $O^*(2^{O(k \log k)})$.

But this is not optimal: LONG CYCLE can be solved in time $O^*(2^{O(k)})$.

LONG CYCLE

Theorem: k-CYCLE can be solved in time $O^*(2^{O(k)})$.

(k-CYCLE: Is there a cycle of length exactly k in G?)

Theorem: LONG CYCLE can be solved in time $O^*(2^{O(k)})$.

(LONG CYCLE: Is there a cycle of length at least k in G?)

LONG CYCLE

Theorem: k-CYCLE can be solved in time $O^*(2^{O(k)})$.
(k-CYCLE: Is there a cycle of length exactly k in G?)

Theorem: LONG CYCLE can be solved in time $O^*(2^{O(k)})$.
(LONG CYCLE: Is there a cycle of length at least k in G?)

Proof:

- Use the k-CYCLE algorithm to test for a cycle of length k, $k+1$, ... , $2k$.
- If no such cycle is found, contract an arbitrary edge, and repeat.
- Stop when a long cycle is found or the graph has no edge.
- Correctness:
  - Any long cycle in the contracted graph is a long cycle of the original.
  - If there is a cycle of length $> 2k$, then there is a cycle of length $\geq k$ after contracting any edge.

LONG CYCLE

Can we solve LONG CYCLE in time $O^*(2^{O(k)})$ using treewidth reduction + treewidth algorithm?

- Can we solve LONG CYCLE in time $O^*(2^{o(w \log w)})$, where w is the treewidth?
- Can we reduce in polynomial time LONG CYCLE to an instance with treewidth $o(k)$?

LONG CYCLE

Can we solve LONG CYCLE in time $O^*(2^{O(k)})$ using treewidth reduction + treewidth algorithm?

- Can we solve LONG CYCLE in time $O^*(2^{o(w \log w)})$, where w is the treewidth?
- Can we reduce in polynomial time LONG CYCLE to an instance with treewidth $o(k)$?
  - Variant 1: The cycle size in the new instance remains the same.
  - Variant 2: The cycle size in the new instance can be arbitrary large.

LONG CYCLE

Can we solve LONG CYCLE in time $O^*(2^{O(k)})$ using treewidth reduction + treewidth algorithm?

- Can we solve LONG CYCLE in time $O^*(2^{o(w \log w)})$, where w is the treewidth?
- Can we reduce in polynomial time LONG CYCLE to an instance with treewidth $o(k)$?
  - Variant 1: The cycle size in the new instance remains the same.
  - Variant 2: The cycle size in the new instance can be arbitrary large.

We might be able to answer questions like that.

Theorem: [Lokshtanov, M, Saurabh: "Slightly Superexponential Parameterized Problems"] Assuming ETH, there is no $O^*(2^{o(w \log w)})$ time algorithm for DISJOINT PATHS.

d-HITTING SET

d-HITTING SET: Given sets of size at most d, can we hit all of them with k elements?

Can be solved in time $O^*(d^k)$ by branching.

Can we match this running time by treewidth reduction + treewidth algorithm?

d-HITTING SET

d-HITTING SET: Given sets of size at most d, can we hit all of them with k elements?

Can be solved in time $O^*(d^k)$ by branching.

Can we match this running time by treewidth reduction + treewidth algorithm?

- Can be solved in time $O^*(2^w)$ if treewidth is at most w and there is no $O^*(2^{o(w)})$ time algorithm (unless ETH fails).
- Can we reduce treewidth to $O(k \log d)$ in polynomial time?

d-HITTING SET

d-HITTING SET: Given sets of size at most d, can we hit all of them with k elements?

Can be solved in time $O^*(d^k)$ by branching.

Can we match this running time by treewidth reduction + treewidth algorithm?

- Can be solved in time $O^*(2^w)$ if treewidth is at most w and there is no $O^*(2^{o(w)})$ time algorithm (unless ETH fails).
- Can we reduce treewidth to $O(k \log d)$ in polynomial time?

There is a kernel with $O(k^{d-1})$ vertices [Abu-Khzam 2010].

$\Rightarrow$ Treewidth can be reduced to $O(k^{d-1})$ in polynomial time. Can we reduce treewidth much better than that?

d-SET PACKING

d-SET PACKING: Given sets of size at most d, can we find k pairwise independent sets?

- Can be solved in time $O^*(2^{O(kd)})$ (e.g., by color coding).
- Can be solved in time $O^*(2^{O(w)})$, where w is treewidth.
- Has a kernel of size $O(k^{d-1})$ vertices [Abu-Khzam 2010].

Can we match the $2^{O(kd)}$ running time by reducing treewidth to $O(kd)$ in polynomial time? Can we get treewidth less than $O(k^{d-1})$?

Bidimensionality
PLANAR k-DOMINATING SET

Kernelization:

- Has a linear-vertex kernel.
- Kernelization + brute foce gives $O^*(2^{O(k)})$ algorithm.

Bidimensionality
PLANAR k-DOMINATING SET

Kernelization:

- Has a linear-vertex kernel.
- Kernelization + brute foce gives $O^*(2^{O(k)})$ algorithm.

Exact algorithm:

- Treewidth of a planar graph is at most $O(\sqrt{n})$.
- Has an $2^{O(\sqrt{n})}$ exact algorithm (and no $2^{o(\sqrt{n})}$ assuming ETH).
- Kernelization + exact algorithm gives the optimal $O^*(2^{O(\sqrt{k})})$ time.

Bidimensionality
PLANAR k-DOMINATING SET

Kernelization:

- Has a linear-vertex kernel.
- Kernelization + brute foce gives $O^*(2^{O(k)})$ algorithm.

Exact algorithm:

- Treewidth of a planar graph is at most $O(\sqrt{n})$.
- Has an $2^{O(\sqrt{n})}$ exact algorithm (and no $2^{o(\sqrt{n})}$ assuming ETH).
- Kernelization + exact algorithm gives the optimal $O^*(2^{O(\sqrt{k})})$ time.

Treewidth reduction:

- The answer is no, unless treewidth is $O(\sqrt{k}) \Rightarrow$ We can reduce treewidth to $O(\sqrt{k})$ in polynomial time.
- Treewidth reduction + treewidth algorithm gives $O^*(2^{O(\sqrt{k})})$ running time.

Bidimensionality
PLANAR k-DOMINATING SET

- Kernelization + brute foce: $O^*(2^{O(k)})$ time.
- Kernelization + exact algorithm: $O^*(2^{O(\sqrt{k})})$ time.
- Treewidth reduction + treewidth algorithm: $O^*(2^{O(\sqrt{k})})$ time.

Summary

- **The meta question:** Which techniques are efficient for which problems?
- Kernelization is not the right technique for some problems.
- Formalization of solving a problem by branching rules.
- Challenging questions: which problems can be solved by branching rules?
- Treewidth reduction.
- What are the problems for which treewidth reduction is competitive with other ideas?