Fixed Parameter Algorithms

Venkatesh Raman

The Institute of Mathematical Sciences, Chennai

Jan 7, 2011, Psgtech, Coimbatore, IGGA, India

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, matching, minimum spanning tree, 2SAT, convext hull, planar drawing, linear programming, etc.
- It is unlikely that polynomial-time algorithms exist for **NP-hard** problems.
- 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

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

In other words: we do not want to be efficient on all inputs of size n, only for those where k is small.

What can be the parameter k?

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

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

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

Main definition:

> A parameterized problem is **fixed-parameter tractable (FPT)** if there is an $f(k)n^c$ time algorithm for some constant c.

Example: VERTEX COVER parameterized by the required size k is FPT:
It is known that it 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.
- ...

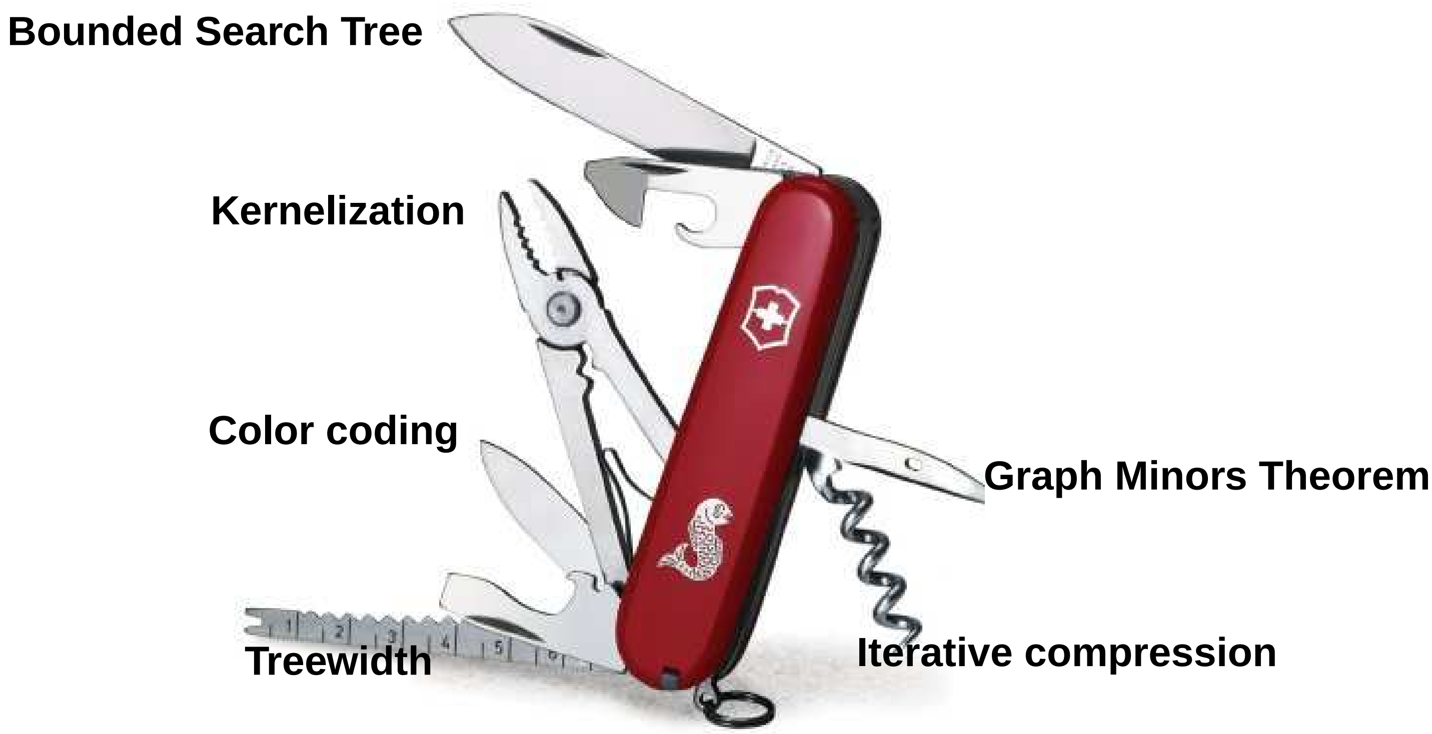
The Birth of Parameterized Complexity

Motivated by Graph Minor Theory of Robertson and Seymour, and notions of treewidth, around 1990-1991, over a series of papers, Downey and Fellows

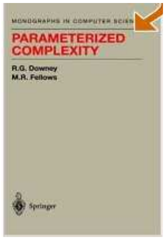
- defined the notion of fixed parameter tractability
- developed hardness theory ($W[1]$, $W[2]$-complete problems),
- classified several problems ‘hard’ and ‘easy’ in this framework, (Eg: Vertex Cover, feedback vertex cover – FPT; Dominating set, weight k satisfying assignment – W-hard)
- applied the framework in several application areas (databases, coding theory, biology, ...)
- built a lot of communities including at IMSc (Fellows!)

FPT algorithmic techniques

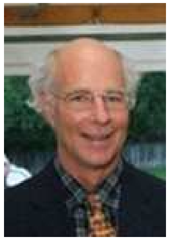
- Significant advances in the past 20 years or so (especially in recent years).
- Powerful toolbox for designing FPT algorithms:

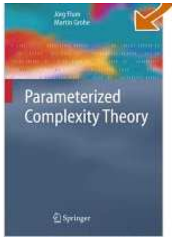


Books

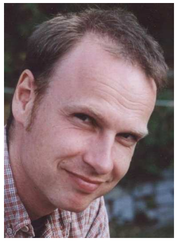


Downey-Fellows: Parameterized Complexity, Springer, 1999





Flum-Grohe: Parameterized Complexity Theory, Springer, 2006



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

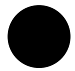
Outline of the talk

- Algorithmic Techniques
  - Bounded Search Trees
  - Kernalization
  - Iterative Compression
  - Color Coding
- Hardness Theory
- Parameterized Complexity and Approximation
- Conclusions

Bounded search tree method

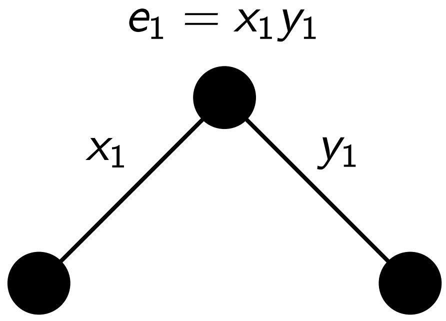
Algorithm for k- VERTEX COVER:

$e_1 = x_1 y_1$



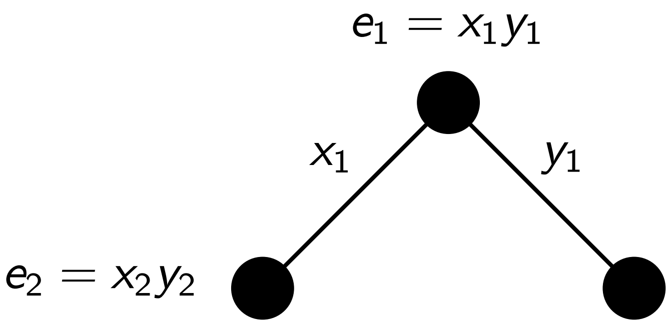
Bounded search tree method

Algorithm for k- VERTEX COVER:



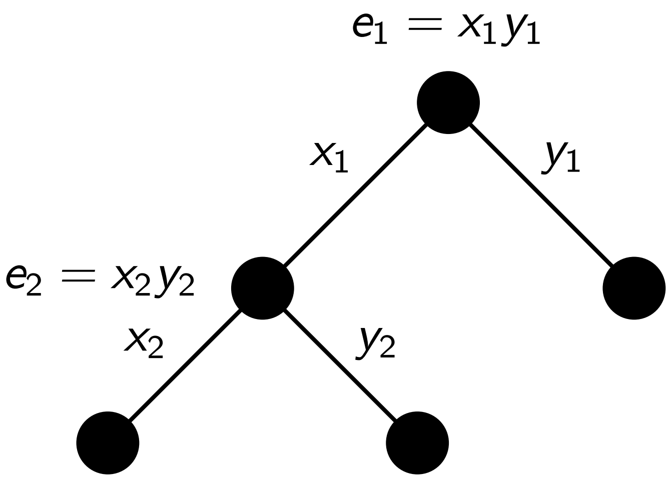
Bounded search tree method

Algorithm for k- VERTEX COVER:



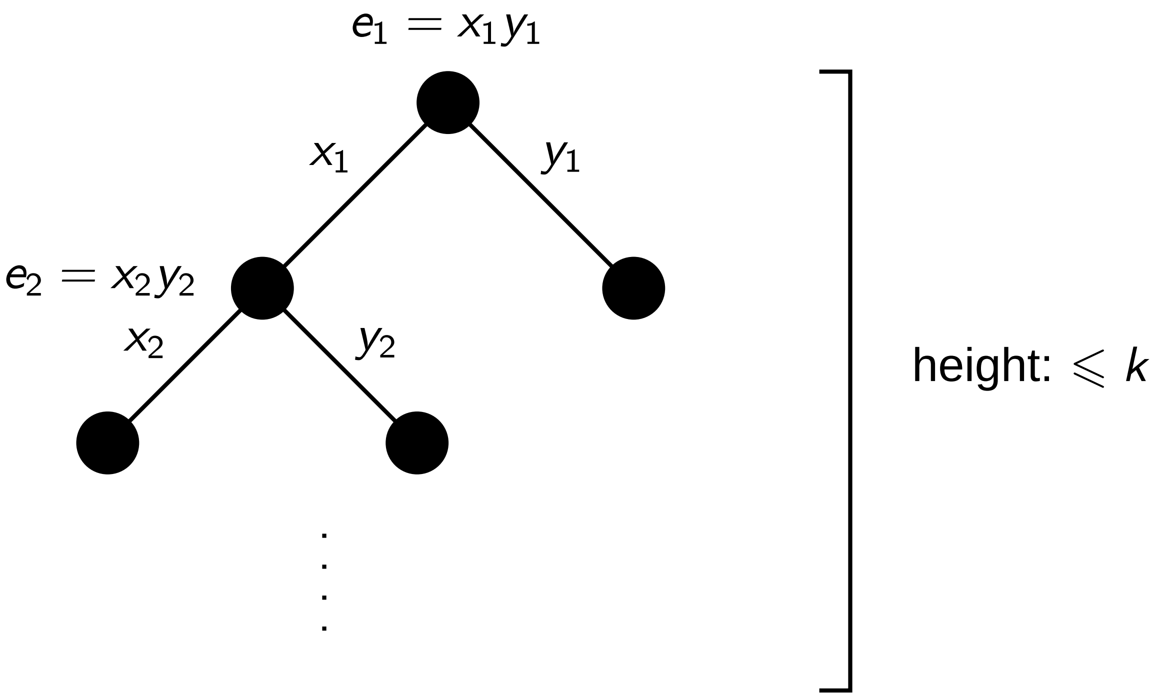
Bounded search tree method

Algorithm for k- VERTEX COVER:



Bounded search tree method

Algorithm for k- VERTEX COVER:



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

Improved Branching Algorithms

Observation: For any vertex x, if x is not in the vertex cover, all its neighbors must be in the vertex cover.

For any vertex x of degree at least 2, check recursively whether

- $G - x$ has a vertex cover of size at most $k - 1$ or
- $G - N(x)$ has a vertex cover of size at most $k - \mathit{degree}(x)$.

If every vertex has degree at most 1, solve in polynomial time.

Improved Branching Algorithms

Observation: For any vertex x, if x is not in the vertex cover, all its neighbors must be in the vertex cover.

For any vertex x of degree at least 2, check recursively whether

- $G - x$ has a vertex cover of size at most $k - 1$ or
- $G - N(x)$ has a vertex cover of size at most $k - \mathit{degree}(x)$.

If every vertex has degree at most 1, solve in polynomial time.

$$T(k) \leqslant T(k-1) + T(k-2)$$

Improved Branching Algorithms

Observation: For any vertex x, if x is not in the vertex cover, all its neighbors must be in the vertex cover.

For any vertex x of degree at least 2, check recursively whether

- $G - x$ has a vertex cover of size at most $k - 1$ or
- $G - N(x)$ has a vertex cover of size at most $k - \mathit{degree}(x)$.

If every vertex has degree at most 1, solve in polynomial time.

$$T(k) \leqslant T(k-1) + T(k-2)$$

Fibonacci recurrence on k that results in $O((1.618)^k m)$

Improved Branching Algorithms

Observation: For any vertex x, if x is not in the vertex cover, all its neighbors must be in the vertex cover.

For any vertex x of degree at least 2, check recursively whether

- $G - x$ has a vertex cover of size at most $k - 1$ or
- $G - N(x)$ has a vertex cover of size at most $k - \mathit{degree}(x)$.

If every vertex has degree at most 1, solve in polynomial time.

$$T(k) \leqslant T(k-1) + T(k-2)$$

Fibonacci recurrence on k that results in $O((1.618)^k m)$

- Can be improved by branching on larger structures and doing a lot of case analyses; the current best is $O(1.28^k + kn)$.
- Technique successfully applied for hitting set, undirected feedback vertex set, directed feedback vertex set in tournaments, maxsat, maxcut, ...

Kernelization

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,
- $k' \leqslant k$, and
- $|I'| \leqslant f(k)$ 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,
- $k' \leqslant k$, and
- $|I'| \leqslant f(k)$ 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.

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,
- $k' \leqslant k$, and
- $|I'| \leqslant f(k)$ 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 $f(k) \leqslant n$, then solve the instance in time $f(k)n^c \leqslant n^{c+1}$, 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 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, $|V(G)| \leqslant k(k+1)$ 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 $\Rightarrow (G \setminus (u \cup v), k-1)$.

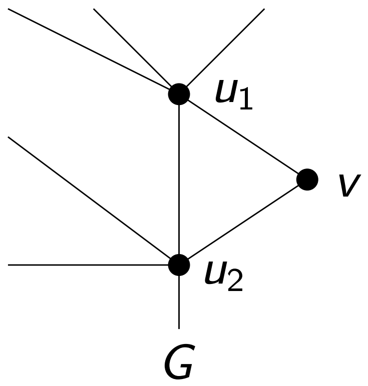
If none of the rules can be applied, then every vertex has degree at least 2.
$\Rightarrow |V(G)| \leqslant |E(G)|$

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

Kernelization for VERTEX COVER

Let us add a fourth rule:

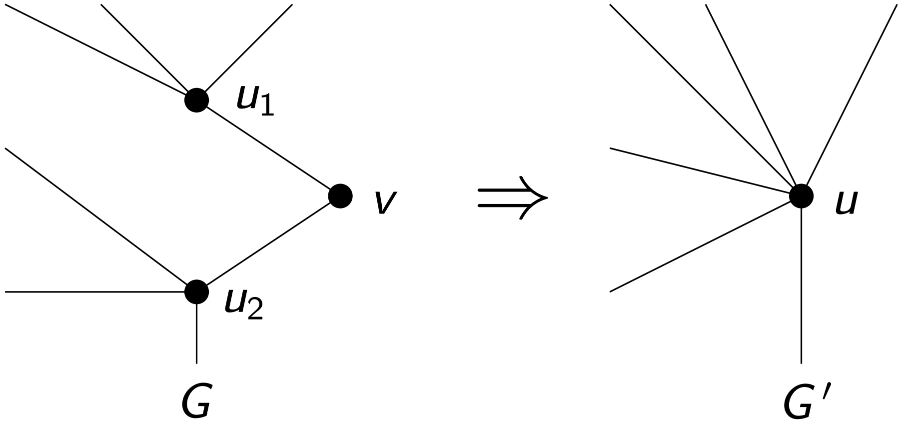
Rule 4a: If v has degree 2, and its neighbors u_1 and u_2 are adjacent, then we can assume that u_1, u_2 are in the solution $\Rightarrow (G \setminus \{u_1, u_2, v\}, k-2)$.



Kernelization for **VERTEX COVER**

Let us add a fourth rule:

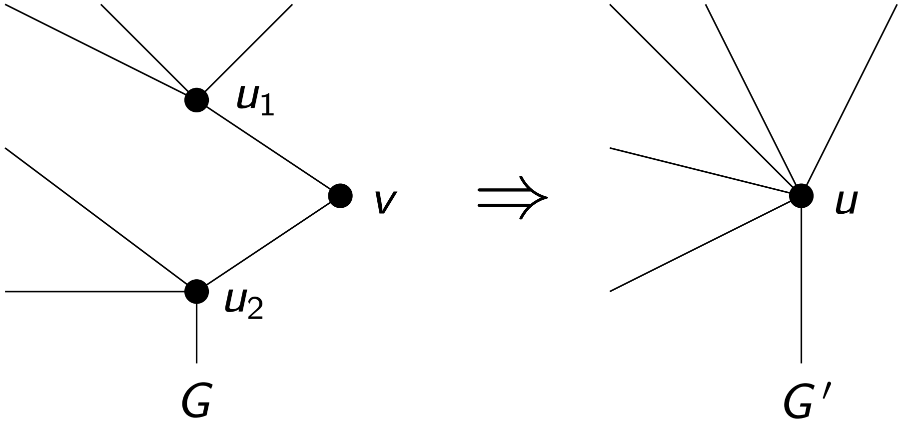
Rule 4b: If v has degree 2, then G' is obtained by identifying the two neighbors of v and deleting $v \Rightarrow (G', k-1)$.



Kernelization for VERTEX COVER

Let us add a fourth rule:

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:

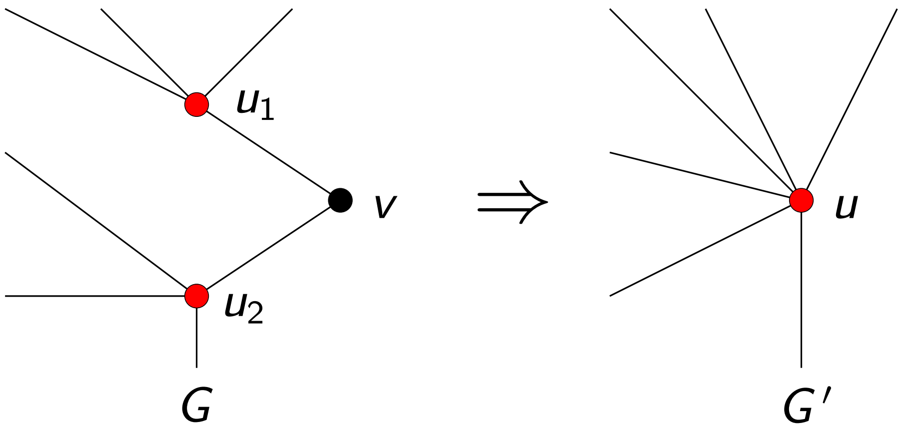
Let S' be a vertex cover of size $k-1$ for G'.

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

Let us add a fourth rule:

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'.

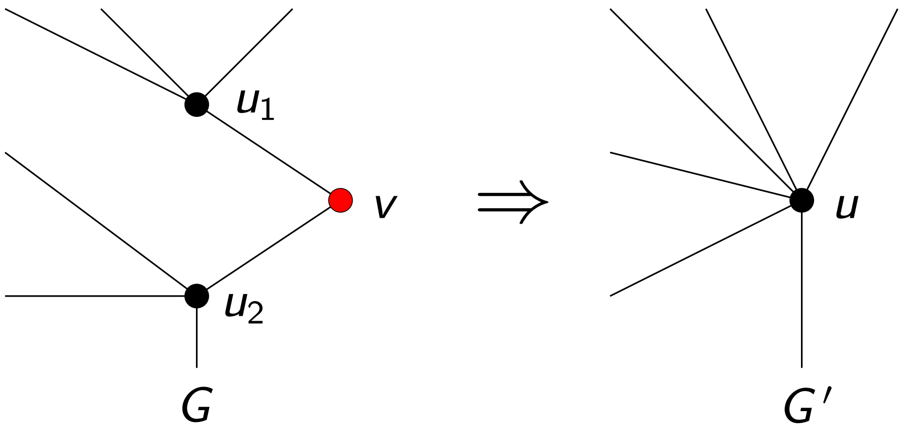
If $u \in S \Rightarrow (S' \setminus u) \cup \{u_1, u_2\}$ is a vertex cover of size k for G.

If $u \notin S \Rightarrow S' \cup v$ is a vertex cover of size k for G.

Kernelization for Vertex Cover

Let us add a fourth rule:

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'.

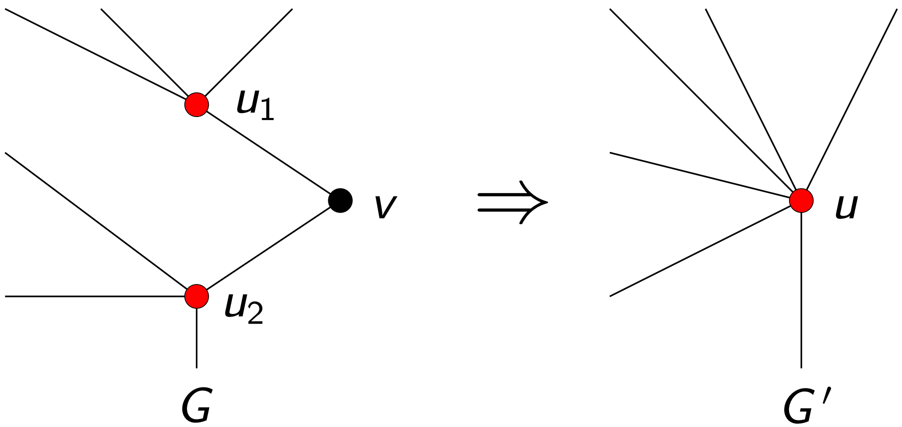
If $u \in S \Rightarrow (S' \setminus u) \cup \{u_1, u_2\}$ is a vertex cover of size k for G.

If $u \notin S \Rightarrow S' \cup v$ is a vertex cover of size k for G.

Kernelization for VERTEX COVER

Let us add a fourth rule:

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'.

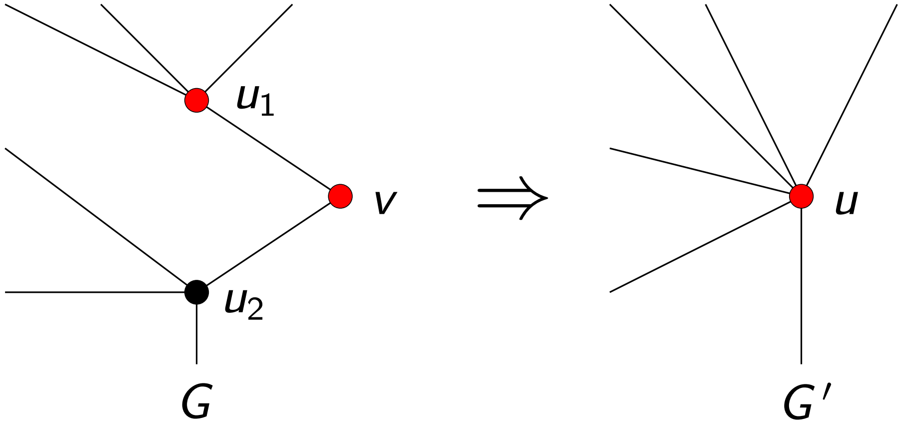
If exactly one of u_1 and u_2 is in S, then $v \in S \Rightarrow (S \setminus \{u_1, u_2, v\}) \cup u$ is a vertex cover of size $k-1$ for G'.

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

Let us add a fourth rule:

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'.

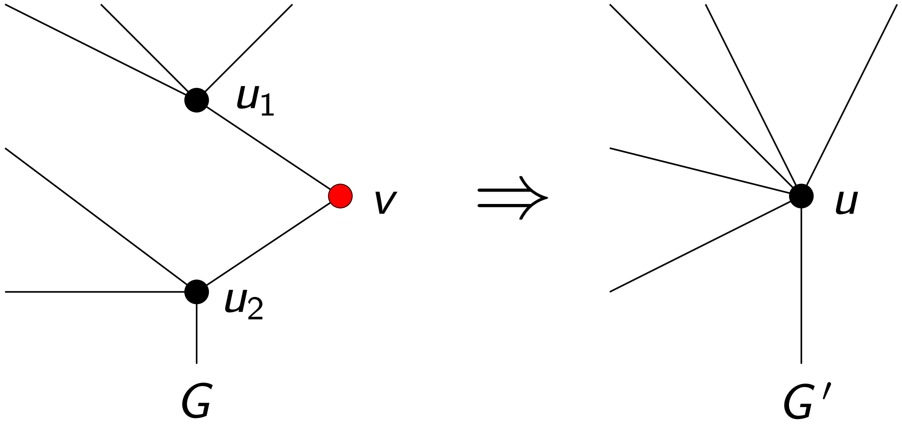
If exactly one of u_1 and u_2 is in S, then $v \in S \Rightarrow (S \setminus \{u_1, u_2, v\}) \cup u$ is a vertex cover of size $k-1$ for G'.

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

Let us add a fourth rule:

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:

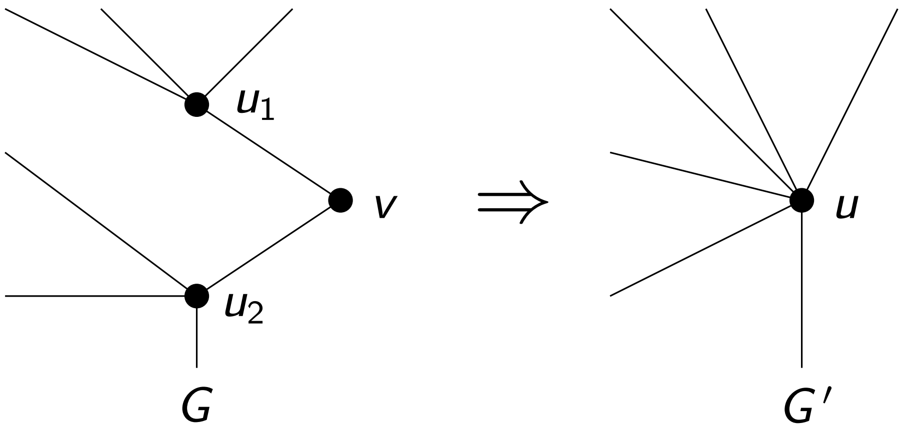
Let S be a vertex cover of size k for G.

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

Let us add a fourth rule:

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, $|V(G)| \leqslant 2|E(G)|/3 \leqslant \frac{2}{3}k^2$ and we have a $\frac{2}{3}k^2$ vertex kernel.

More on kernels

- There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.
- There is an $O(k^2)$ kernel for undirected feedback vertex set (SODA 2009) – uses Hall's like theorem.
- Linear kernel for dominating set in Planar graphs (Alber et al JACM 2004); generalized for more parameters in larger classes of graphs (BFLPS in FOCS 2010)

More on kernels

- There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.
- There is an $O(k^2)$ kernel for undirected feedback vertex set (SODA 2009) – uses Hall's like theorem.
- Linear kernel for dominating set in Planar graphs (Alber et al JACM 2004); generalized for more parameters in larger classes of graphs (BFLPS in FOCS 2010)

Famous Open problems: Polynomial sized ($k^{O(1)}$) kernel for

- Directed feedback vertex set?
- Odd cycle transversal (set of vertices whose removal results in a bipartite graph)?

Kernelization

- 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… tricks (Crown Reduction and the Sunflower Lemma).
- Recently this topic got a lot of attention due to recent machineries that show lower bounds on kernel sizes (i.e. no polynomial size kernel or $O(k)$ kernel possible under complexity theoretic assumptions).

Iterative Compression

- A powerful technique (for minimization problems)
- Given a solution of size $k+1$, check whether there is one of size k; This is the compression step; somehow starting with a solution helps.
- How do we get the given $k+1$-sized solution? We iterate and compress!

Iterative Compression

- A powerful technique (for minimization problems)
- Given a solution of size $k+1$, check whether there is one of size k; This is the compression step; somehow starting with a solution helps.
- How do we get the given $k+1$-sized solution? We iterate and compress!

The first $k+1$ vertices of the graph is a solution for the graph induced on that set of vertices.

- Compress if possible; if not possible, say NO.
  For, if the induced subgraph has no k-sized solution, the original graph can not have.
- If compressible, expand the compressed solution to get a solution for the graph induced on one more vertex to get a $k+1$-sized solution for a larger graph.

Overall time is $O((n-k) * \mathit{time\ for\ compression\ step})$.

More on Iterative Compression

Several recent results were shown FPT using iterative compression

1. Directed Feedback Vertex Set (STOC 08, JACM 2009)
2. Within k clauses from 2SAT (ICALP 08)
3. Cochromatic Number in perfect graphs (SWAT 2010)
4. Odd Cycle Transversal (the first one, in ORL)
5. Multicut problems

Color coding for finding k-path

- A randomized technique (Alon, Yuster, Zwick JACM 95)
- **Problem:** Is there a simple path of length k (or more) in G?
- NP-complete as this is a decision version of Hamiltonian path.

Color coding for finding k-path

- A randomized technique (Alon, Yuster, Zwick JACM 95)
- **Problem:** Is there a simple path of length k (or more) in G?
- NP-complete as this is a decision version of Hamiltonian path.

Color Coding Algorithm

1. Randomly color the vertices of the graph with integers 1 to k.
2. Find a colorful path (a path where all colors are distinct) of length k if exists (using Dynamic Programming, can have a start vertex. Remember color sets of size i ($\binom{k}{i}$) in paths of length i at intermediate steps. $O(2^k m)$)
3. Else repeat

Color coding for finding k-path

- A randomized technique (Alon, Yuster, Zwick JACM 95)
- **Problem:** Is there a simple path of length k (or more) in G?
- NP-complete as this is a decision version of Hamiltonian path.

Color Coding Algorithm

1. Randomly color the vertices of the graph with integers 1 to k.
2. Find a colorful path (a path where all colors are distinct) of length k if exists (using Dynamic Programming, can have a start vertex. Remember color sets of size i ($\binom{k}{i}$) in paths of length i at intermediate steps. $O(2^k m)$)
3. Else repeat

If there is a simple path of length k, it will be colorful with probability $k!/k^k$ which is $\Omega(e^{-k})$. So, expected # of repetitions – $O(e^k)$.
Can be derandomized using perfect hash families.

More on Color Coding

1. Can find k-path, k-cycle, k-tree, subgraphs of bounded treewidth with k vertices all in FPT time.
2. Chromatic Coding – a generalization applied to get a $2^{O(\sqrt{k}\log k)} + n^{O(1)}$ algorithm for finding Feedback Arc Set in tournaments (ALS ICALP 2009).

Hardness

- Parameterized Reductions (converts (x, k) to (x', k') where k' is a function of k, and the runtime takes $g(k)n^{O(1)}$.
- W-hardness theory (W-hard implies unlikely to have $f(k)n^{O(1)}$ algorithm)
- Independent Set, Clique, Weight k satisfying assignment in a bounded CNF formula, hard for $W[1]$.
- Dominating Set, Set Cover – hard for $W[2]$.

Hardness

- Parameterized Reductions (converts (x, k) to (x', k') where k' is a function of k, and the runtime takes $g(k)n^{O(1)}$.
- W-hardness theory (W-hard implies unlikely to have $f(k)n^{O(1)}$ algorithm)
- Independent Set, Clique, Weight k satisfying assignment in a bounded CNF formula, hard for $W[1]$.
- Dominating Set, Set Cover – hard for $W[2]$.
- Recent Lower bounds on Kernels (Recall that FPT = Kernelizable) give finer classification
- Under Exponential Time Hypothesis (SAT has no $2^{o(n)}$ algorithm), there are some lower bounds for the $f(k)$ functions known.

Hardness

- Parameterized Reductions (converts (x, k) to (x', k') where k' is a function of k, and the runtime takes $g(k)n^{O(1)}$.
- W-hardness theory (W-hard implies unlikely to have $f(k)n^{O(1)}$ algorithm)
- Independent Set, Clique, Weight k satisfying assignment in a bounded CNF formula, hard for $W[1]$.
- Dominating Set, Set Cover – hard for $W[2]$.
- Recent Lower bounds on Kernels (Recall that FPT = Kernelizable) give finer classification
- Under Exponential Time Hypothesis (SAT has no $2^{o(n)}$ algorithm), there are some lower bounds for the $f(k)$ functions known.

Hardness

- Parameterized Reductions (converts (x, k) to (x', k') where k' is a function of k, and the runtime takes $g(k)n^{O(1)}$.
- W-hardness theory (W-hard implies unlikely to have $f(k)n^{O(1)}$ algorithm)
- Independent Set, Clique, Weight k satisfying assignment in a bounded CNF formula, hard for $W[1]$.
- Dominating Set, Set Cover – hard for $W[2]$.
- Recent Lower bounds on Kernels (Recall that FPT = Kernelizable) give finer classification
- Under Exponential Time Hypothesis (SAT has no $2^{o(n)}$ algorithm), there are some lower bounds for the $f(k)$ functions known.

Hardness

- Parameterized Reductions (converts (x, k) to (x', k') where k' is a function of k, and the runtime takes $g(k)n^{O(1)}$.
- W-hardness theory (W-hard implies unlikely to have $f(k)n^{O(1)}$ algorithm)
- Independent Set, Clique, Weight k satisfying assignment in a bounded CNF formula, hard for $W[1]$.
- Dominating Set, Set Cover – hard for $W[2]$.
- Recent Lower bounds on Kernels (Recall that FPT = Kernelizable) give finer classification
- Under Exponential Time Hypothesis (SAT has no $2^{o(n)}$ algorithm), there are some lower bounds for the $f(k)$ functions known.

Approximation and Parameterized Complexity

- The parameterized version of every MaxSNP, MinF+ problem is in FPT.
- There are easy to approximate problems whose decision versions are W-hard (rectangle stabbing) and
- there are FPT problems (k-path, odd cycle traversal) whose optimization versions are hard to approximate.
- MaxSNP hard problems can not have subexponential parameterized problems unless ETH is false.
- A large class of bidimensional parameters have EPTAS in a large class of graphs (FLRS 2011).

Conclusions

- Matured as a serious paradigm with a host of toolkits for algorithms and hardness
- Continues to make dents in application areas
- Finer classifications,
  - in the size of the kernels for easy problems,
  - in the running time for harder problems
- new connections (say, to approximation), and
- new algorithmic techniques and new parameterizations

continue to be discovered.

Concrete Open Problems

1. Does G have a $K_{k,k}$? FPT or W-hard?
2. Polynomial kernels for DFVS, OCT, ...
3. Does a planar graph have an independent set of size at least $n/4 + k$? FPT or W-hard?

References

1. Invitation to Fixed-Parameter Algorithms – Rolf Niedermeir (Oxford UP 2006)
2. Parameterized Complexity – Rod Downey and Mike Fellows (Springer 1999)
3. Parameterized Complexity Theory – Jörg Flum and Martin Grohe (Springer 2006)
4. Proceedings of IPEC, and other conferences

Thank You