Fixed-Parameter Algorithms

Venkatesh Raman

Theoretical Computer Science group
The Institute of Mathematical Sciences
Chennai

January 8, 2010
Talk at NIT Tiruchi

Outline

Introduction – The paradigm and motivation

Introduction – The paradigm and motivation

- A theoretically rich and practically useful paradigm to deal with NP-completeness

Introduction – The paradigm and motivation

- A theoretically rich and practically useful paradigm to deal with NP-completeness
- A framework to do refined, *multidimensional* algorithmic complexity analysis.

Introduction – The paradigm and motivation

- A theoretically rich and practically useful paradigm to deal with NP-completeness
- A framework to do refined, *multidimensional* algorithmic complexity analysis.
- Associated with the input instance x, is given a *parameter* k; The running time is analyzed in terms of *both* the input size and the parameter.

Introduction – The paradigm and motivation

- A theoretically rich and practically useful paradigm to deal with NP-completeness
- A framework to do refined, *multidimensional* algorithmic complexity analysis.
- Associated with the input instance x, is given a *parameter* k; The running time is analyzed in terms of *both* the input size and the parameter.

  Confine the combinatorial explosion to parameters that are likely to be small.

Introduction – The paradigm and motivation

- A theoretically rich and practically useful paradigm to deal with NP-completeness
- A framework to do refined, *multidimensional* algorithmic complexity analysis.
- Associated with the input instance x, is given a *parameter* k;
  The running time is analyzed in terms of *both* the input size and the parameter.
  Confine the combinatorial explosion to parameters that are likely to be small.
  The aim is to design algorithms that take $O(f(k)|x|^{O(1)})$ time where f is a (moderately) exponential function.

Introduction – The paradigm and motivation Cont'd

- There are more than one ways to parameterize a problem; Natural parameters are: the solution size, the size of the alphabet, dimension of the points, the maximum degree, the treewidth, the pathwidth, ...

Introduction – The paradigm and motivation Cont'd

- There are more than one ways to parameterize a problem; Natural parameters are: the solution size, the size of the alphabet, dimension of the points, the maximum degree, the treewidth, the pathwidth, ...
- A problem may be hard when parameterized by one parameter, but easy with respect to another.

Introduction – The paradigm and motivation Cont'd

- There are more than one ways to parameterize a problem; Natural parameters are: the solution size, the size of the alphabet, dimension of the points, the maximum degree, the treewidth, the pathwidth, ...
- A problem may be hard when parameterized by one parameter, but easy with respect to another.
- Active area of research, there are sessions on this topic in major conferences, there are specialized conferences – IPEC (international symposium on parameterized and exact computation; 2010 event in Chennai).

Definitions

- A parameterized language $L \subseteq \Sigma^* \times N$ where Σ is a finite alphabet.

Definitions

- A parameterized language $L \subseteq \Sigma^* \times N$ where Σ is a finite alphabet.
- A parameterized language (YES instances of a decision problem) is *fixed-parameter tractable* (FPT) if there is an algorithm that can decide for any input (x, k), whether it is in the language in time $f(k)|x|^{O(1)}$ time where f is any function of k;
- FPT algorithms (with moderately growing $f(k)$) are useful in practice when the parameter k is small; and there are areas where small parameters capture most practical instances.

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n|x|^{O(1)}$ algorithm.

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n|x|^{O(1)}$ algorithm.
2. $L = \{\textit{Triangle free graphs with a } k - \textit{independent set}\}$
   (i.e. Does a given triangle free graph have an independent set of size at least k?)

Examples

1. $L = \{F | F \text{ is a satisfiable boolean CNF formula on } n \text{ variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n|x|^{O(1)}$ algorithm.
2. $L = \{\text{Triangle free graphs with a } k - \text{independent set}\}$
   (i.e. Does a given triangle free graph have an independent set of size at least k?)
   Parameter: k

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n |x|^{O(1)}$ algorithm.
2. $L = \{\textit{Triangle free graphs with a } k - \textit{independent set}\}$
   (i.e. Does a given triangle free graph have an independent set of size at least k?)
   Parameter: k
   There is an easy $O(n^k k^2)$ algorithm, but this runtime is not of the form $f(k)n^{O(1)}$.

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n|x|^{O(1)}$ algorithm.
2. $L = \{\textit{Triangle free graphs with a } k - \textit{independent set}\}$
   (i.e. Does a given triangle free graph have an independent set of size at least k?)
   Parameter: k
   There is an easy $O(n^k k^2)$ algorithm, but this runtime is not of the form $f(k)n^{O(1)}$. But L is FPT by a $O(k^{2k} + n^{O(1)})$ algorithm as described below.

Examples

1. $L = \{F | F \textit{ is a satisfiable boolean CNF formula on n variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n|x|^{O(1)}$ algorithm.
2. $L = \{\textit{Triangle free graphs with a } k - \textit{independent set}\}$
   (i.e. Does a given triangle free graph have an independent set of size at least k?)
   Parameter: k
   There is an easy $O(n^k k^2)$ algorithm, but this runtime is not of the form $f(k)n^{O(1)}$. But L is FPT by a $O(k^{2k} + n^{O(1)})$ algorithm as described below.
   If $n > R(3, k)$ (the Ramsey number), then $G \in L$

Examples

1. $L = \{F | F \text{ is a satisfiable boolean CNF formula on } n \text{ variables}\}$
   (i.e. Is the given formula on n variables satisfiable?)
   Parameter: n
   The problem is FPT by a $2^n|x|^{O(1)}$ algorithm.
2. $L = \{\text{Triangle free graphs with a } k-\text{independent set}\}$
   (i.e. Does a given triangle free graph have an independent set of size at least k?)
   Parameter: k
   There is an easy $O(n^k k^2)$ algorithm, but this runtime is not of the form $f(k)n^{O(1)}$. But L is FPT by a $O(k^{2k} + n^{O(1)})$ algorithm as described below.
   If $n > R(3, k)$ (the Ramsey number), then $G \in L$
   else ($n \leq R(3, k) \leq k^2$) check whether G has an independent set of size k by trying all subsets of size at most k.

Examples, Continued

Example 3:
$L = \{Planar\ Graphs\ with\ a\ k - dominating\ set\}$
(i.e. Does a given planar graph G have a dominating set of size at most k?)

Examples, Continued

Example 3:

$L = \{Planar\ Graphs\ with\ a\ k-dominating\ set\}$

(i.e. Does a given planar graph G have a dominating set of size at most k?)

Parameter: k

Examples, Continued

Example 3:
$L = \{Planar\ Graphs\ with\ a\ k - dominating\ set\}$
(i.e. Does a given planar graph G have a dominating set of size at most k?)
Parameter: k
There is an easy $O(n^{k+O(1)})$ algorithm again!

Examples, Continued

Example 3:
$L = \{Planar\ Graphs\ with\ a\ k - dominating\ set\}$
(i.e. Does a given planar graph G have a dominating set of size at most k?)
Parameter: k
There is an easy $O(n^{k+O(1)})$ algorithm again! L is FPT by an $2^{O(\sqrt{k})} + n^{O(1)}$ algorithm.

Examples, Continued

Example 3:
$L = \{Planar\ Graphs\ with\ a\ k - dominating\ set\}$
(i.e. Does a given planar graph G have a dominating set of size at most k?)
Parameter: k
There is an easy $O(n^{k+O(1)})$ algorithm again! L is FPT by an $2^{O(\sqrt{k})} + n^{O(1)}$ algorithm.
Domination number is a contraction closed bidimensional parameter, and hence treewidth of a planar graph with a k-dominating set is $O(\sqrt{k})$, apply dynamic programming.

Outline

Historical Perspective –Graph Minor Theorems

1. Robertson-Seymour's theorem on Wagner's conjecture (GMTheorem 1): Every *minor closed* family of finite graphs has a finite *forbidden set*.

Historical Perspective –Graph Minor Theorems

1. Robertson-Seymour's theorem on Wagner's conjecture (GMTheorem 1): Every *minor closed* family of finite graphs has a finite *forbidden set*.
2. Minor of a graph – graph obtained by contracting some edges and deleting some edges.

Historical Perspective –Graph Minor Theorems

1. Robertson-Seymour's theorem on Wagner's conjecture (GMTheorem 1): Every *minor closed* family of finite graphs has a finite *forbidden set*.
2. Minor of a graph – graph obtained by contracting some edges and deleting some edges.
3. (RS: Theorem 2) Given a graph G on n vertices, and a (fixed) graph H, we can check in $O(f(|H|)n^3)$ time whether H is a minor of G.

Historical Perspective –Graph Minor Theorems

1. Robertson-Seymour's theorem on Wagner's conjecture (GMTheorem 1): Every *minor closed* family of finite graphs has a finite *forbidden set*.
2. Minor of a graph – graph obtained by contracting some edges and deleting some edges.
3. (RS: Theorem 2) Given a graph G on n vertices, and a (fixed) graph H, we can check in $O(f(|H|)n^3)$ time whether H is a minor of G.
4. For a fixed k, the following families are minor closed.
   $L_k = \{Graphs\ having\ at\ most\ k\ sized\ vertex\ covers(VC)\}$
   $L_k =$
   $\{Graphs\ having\ at\ most\ k\ sized\ feedback\ vertex\ sets(FVS)\}$

Historical Perspective –Graph Minor Theorems

1. Robertson-Seymour's theorem on Wagner's conjecture (GMTheorem 1): Every *minor closed* family of finite graphs has a finite *forbidden set*.
2. Minor of a graph – graph obtained by contracting some edges and deleting some edges.
3. (RS: Theorem 2) Given a graph G on n vertices, and a (fixed) graph H, we can check in $O(f(|H|)n^3)$ time whether H is a minor of G.
4. For a fixed k, the following families are minor closed. $L_k = \{Graphs\ having\ at\ most\ k\ sized\ vertex\ covers(VC)\}$ $L_k = \{Graphs\ having\ at\ most\ k\ sized\ feedback\ vertex\ sets(FVS)\}$ and hence by the above two theorems, checking whether a graph has a VC or FVS of size at most k is FPT.

Historical Perspective –Graph Minor Theorems

1. Robertson-Seymour's theorem on Wagner's conjecture (GMTheorem 1): Every *minor closed* family of finite graphs has a finite *forbidden set*.
2. Minor of a graph – graph obtained by contracting some edges and deleting some edges.
3. (RS: Theorem 2) Given a graph G on n vertices, and a (fixed) graph H, we can check in $O(f(|H|)n^3)$ time whether H is a minor of G.
4. For a fixed k, the following families are minor closed.
   $L_k = \{Graphs\ having\ at\ most\ k\ sized\ vertex\ covers(VC)\}$
   $L_k =$
   $\{Graphs\ having\ at\ most\ k\ sized\ feedback\ vertex\ sets(FVS)\}$
   and hence by the above two theorems, checking whether a graph has a VC or FVS of size at most k is FPT.
5. However the $f()$ had a HUGE towers of exponents and the proof of GMT is also non constructive!

Historical Perspective – Treewidth etc

1. (Bodlaender et al) Vertex Cover, Dominating Set, Independent Set, Feedback Vertex Set are FPT when the *treewidth* is the parameter. I.e. in $O(2^w n^{O(1)})$ time, one can find the minimum vertex cover, maximum independent set etc on graphs of treewidth at most w.

Historical Perspective – Treewidth etc

1. (Bodlaender et al) Vertex Cover, Dominating Set, Independent Set, Feedback Vertex Set are FPT when the *treewidth* is the parameter. I.e. in $O(2^w n^{O(1)})$ time, one can find the minimum vertex cover, maximum independent set etc on graphs of treewidth at most w.
2. (Courcelle) Any property expressible in monadic second order logic has a linear time algorithm on bounded treewidth graphs.
3. Checking whether a (general) graph has a clique or an independent set or a dominating set of size k seems to require $\Omega(n^k)$ time.

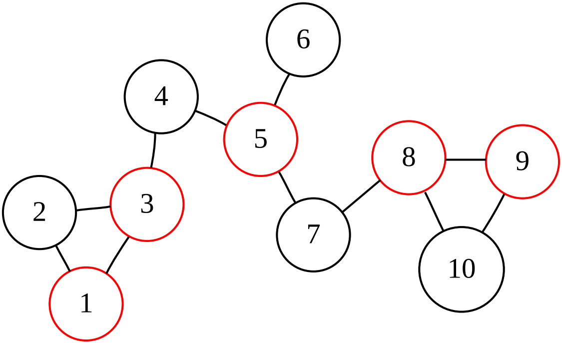
Historical Perspective – Treewidth etc

1. (Bodlaender et al) Vertex Cover, Dominating Set, Independent Set, Feedback Vertex Set are FPT when the *treewidth* is the parameter. I.e. in $O(2^w n^{O(1)})$ time, one can find the minimum vertex cover, maximum independent set etc on graphs of treewidth at most w.
2. (Courcelle) Any property expressible in monadic second order logic has a linear time algorithm on bounded treewidth graphs.
3. Checking whether a (general) graph has a clique or an independent set or a dominating set of size k seems to require $\Omega(n^k)$ time.
4. Downey-Fellows developed hardness theory ($W[1]$, $W[2]$-complete problems) and opened up the area (in late eighties, early nineties).

Outline

Bounded Search Trees – example Vertex Cover

Given a graph $G = (V, E)$, and a parameter k, does G have a vertex cover of size at most k?



Branching Algorithms for k-Vertex Cover

1. Try all subsets of size at most k – $O(n^k m)$.

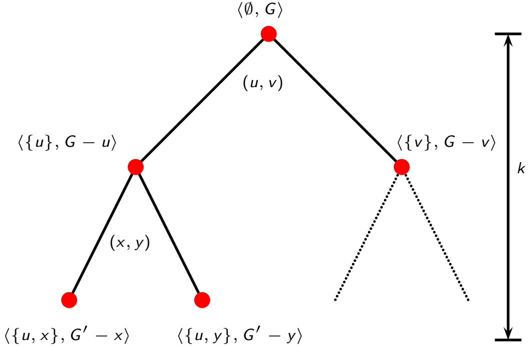
Branching Algorithms for k-Vertex Cover

1. Try all subsets of size at most k – $O(n^k m)$.
2. For every edge (x, y) recursively check whether $G - x$ or $G - y$ has a vertex cover of size at most $k - 1$ recursively.

Branching Algorithms for k-Vertex Cover

1. Try all subsets of size at most k – $O(n^k m)$.
2. For every edge (x, y) recursively check whether $G - x$ or $G - y$ has a vertex cover of size at most $k - 1$ recursively. $O(2^k m)$.

Basic Idea: Given *any* edge (u, v) either u or v is in the solution.



- no. of nodes $\leq 2^k$
- time spent at each node $= O(m)$
- total time taken $= O(2^k \cdot m)$

Improved Branching Algorithms

Improved Branching Algorithms

1. For any vertex x of degree at least 2, check whether
   - $G - x$ has a vertex cover of size at most $k - 1$ or

Improved Branching Algorithms

1. For any vertex x of degree at least 2, check 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 $k - degree(x)$

   recursively.

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

Improved Branching Algorithms

1. For any vertex x of degree at least 2, check 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 $k - degree(x)$

   recursively.

   If every vertex has degree at most 1, solve in polynomial time.
   $T(k) \leq T(k-1) + T(k-2)$

Improved Branching Algorithms

1. For any vertex x of degree at least 2, check 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 $k - degree(x)$

   recursively.

   If every vertex has degree at most 1, solve in polynomial time.
   $T(k) \leq T(k-1) + T(k-2)$ Fibonacci recurrence on k that results in $O((1.618)^k m)$.

Improved Branching Algorithms

1. For any vertex x of degree at least 2, check 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 $k - degree(x)$

   recursively.

   If every vertex has degree at most 1, solve in polynomial time. $T(k) \leq T(k-1) + T(k-2)$ Fibonacci recurrence on k that results in $O((1.618)^k m)$.
2. Can be improved by branching on larger structures and doing a lot of case analyses; the current best is $O(1.27^k + kn)$.

Branching for the Undirected Feedback Vertex Set Problem

Definition

Given an undirected graph $G(V, E)$ and an integer k, are there k vertices whose removal makes G acyclic?

Branching for the Undirected Feedback Vertex Set Problem

Definition

Given an undirected graph $G(V, E)$ and an integer k, are there k vertices whose removal makes G acyclic?

- classical NP-hard problem;

Branching for the Undirected Feedback Vertex Set Problem

Definition

Given an undirected graph $G(V, E)$ and an integer k, are there k vertices whose removal makes G acyclic?

- classical NP-hard problem;
- can be approximated to ratio-2 of optimal;

Branching for the Undirected Feedback Vertex Set Problem

Definition

Given an undirected graph $G(V, E)$ and an integer k, are there k vertices whose removal makes G acyclic?

- classical NP-hard problem;
- can be approximated to ratio-2 of optimal;
- For each subset $S \subseteq V$ of size at most k check whether $G - S$ is acyclic.

Branching for the Undirected Feedback Vertex Set Problem

Definition

Given an undirected graph $G(V, E)$ and an integer k, are there k vertices whose removal makes G acyclic?

- classical NP-hard problem;
- can be approximated to ratio-2 of optimal;
- For each subset $S \subseteq V$ of size at most k check whether $G - S$ is acyclic.
- Time complexity: $O(n^k m)$.

Towards an FPT Algorithm

Standard Preprocessing for FVS algorithms:

Towards an FPT Algorithm

Standard Preprocessing for FVS algorithms:

1. Delete degree one vertices.

Towards an FPT Algorithm

Standard Preprocessing for FVS algorithms:

1. Delete degree one vertices.
2. Short circuit degree two vertices.

Towards an FPT Algorithm

Standard Preprocessing for FVS algorithms:

1. Delete degree one vertices.
2. Short circuit degree two vertices.

- Repeat steps 1 and 2 until no longer possible.

Towards an FPT Algorithm

Standard Preprocessing for FVS algorithms:

1. Delete degree one vertices.
2. Short circuit degree two vertices.

- Repeat steps 1 and 2 until no longer possible.
- *Result:* A (multi) graph with minimum degree 3.

Towards an FPT Algorithm

Standard Preprocessing for FVS algorithms:

1. Delete degree one vertices.
2. Short circuit degree two vertices.

- Repeat steps 1 and 2 until no longer possible.
- *Result:* A (multi) graph with minimum degree 3.

Lemma (Erdos and Posa)

An undirected graph on n vertices with minimum degree 3 has a cycle of length $O(\log n)$.

An FPT Algorithm . . .

An FPT Algorithm . . .

1. Preprocess to get minimum degree 3.

An FPT Algorithm . . .

1. Preprocess to get minimum degree 3.
2. Find a shortest cycle C.

An FPT Algorithm . . .

1. Preprocess to get minimum degree 3.
2. Find a shortest cycle C.
3. For every vertex $v \in C$ check whether $G - v$ has an FVS of size at most $k - 1$.

An FPT Algorithm . . .

1. Preprocess to get minimum degree 3.
2. Find a shortest cycle C.
3. For every vertex $v \in C$ check whether $G - v$ has an FVS of size at most $k - 1$.
4. If Step 3 is true for some v, then answer YES (FVS$(G - v) \cup \{v\}$ is an FVS of size k); else answer NO.

An FPT Algorithm . . .

1. Preprocess to get minimum degree 3.
2. Find a shortest cycle C.
3. For every vertex $v \in C$ check whether $G - v$ has an FVS of size at most $k - 1$.
4. If Step 3 is true for some v, then answer YES ($\mathrm{FVS}(G - v) \cup \{v\}$ is an FVS of size k); else answer NO.

- Since C is of length $O(\log n)$, we get an $O((\log n)^k m)$ time algorithm.

An FPT Algorithm ...

1. Preprocess to get minimum degree 3.
2. Find a shortest cycle C.
3. For every vertex $v \in C$ check whether $G - v$ has an FVS of size at most $k - 1$.
4. If Step 3 is true for some v, then answer YES (FVS$(G - v) \cup \{v\}$ is an FVS of size k); else answer NO.

- Since C is of length $O(\log n)$, we get an $O((\log n)^k m)$ time algorithm.
- Since $(\log n)^k \leq k^{2k} + n$, we have an FPT algorithm, where $f(k) = k^{2k}$.

Towards a better $f(k)$

- A generalization of Erdos and Posa:

Lemma

Any graph with minimum degree 3 *and FVS of size at most* $\sqrt{n/2}$*, has a cycle of length at most* 6*.*

  Using this, one can get a bound of $O^*((\log k)^k)$.

- Improving to $O^*(c^k)$, will be done later.

Kernalization - A formal way of analysing *preprocessing strategies*

1. **Algorithmic idea:** Given (x, k) reduce in polynomial (in $|x|, k$) time to an 'equivalent instance' (x', k') such that
   - $(x, k) \in L$ iff $(x', k') \in L$ and
   - $|x'| \leq g(k)$ for some function g of k.

Kernalization - A formal way of analysing *preprocessing strategies*

1. **Algorithmic idea:** Given (x, k) reduce in polynomial (in $|x|, k$) time to an 'equivalent instance' (x', k') such that
   - $(x, k) \in L$ iff $(x', k') \in L$ and
   - $|x'| \leq g(k)$ for some function g of k.
2. Such an algorithm *kernalizes* the problem, and such a problem is said to be *kernalizable*. x' is said to be the kernel and $g(k)$ is the kernel size.

Kernalization - A formal way of analysing *preprocessing strategies*

1. **Algorithmic idea:** Given (x, k) reduce in polynomial (in $|x|, k$) time to an 'equivalent instance' (x', k') such that
   - $(x, k) \in L$ iff $(x', k') \in L$ and
   - $|x'| \leq g(k)$ for some function g of k.
2. Such an algorithm *kernalizes* the problem, and such a problem is said to be *kernalizable*. x' is said to be the kernel and $g(k)$ is the kernel size.
3. Theorem (easy): A parameterized problem is kernalizable implies it is in FPT.

Kernalization - A formal way of analysing *preprocessing strategies*

1. **Algorithmic idea:** Given (x, k) reduce in polynomial (in $|x|, k$) time to an 'equivalent instance' (x', k') such that
   - $(x, k) \in L$ iff $(x', k') \in L$ and
   - $|x'| \leq g(k)$ for some function g of k.
2. Such an algorithm *kernalizes* the problem, and such a problem is said to be *kernalizable*. x' is said to be the kernel and $g(k)$ is the kernel size.
3. Theorem (easy): A parameterized problem is kernalizable implies it is in FPT.
4. Converse is also true: A parameterized problem is FPT implies it is kernalizable; but the kernel size will be exponential in k.

A kernel for Vertex Cover

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).
   Observation: All vertices in S must be in any vertex cover of size at most k.

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).
   Observation: All vertices in S must be in any vertex cover of size at most k.
2. If $|S| > k$, stop and return NO. Else, delete all vertices of S from G to get G'.

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).
   Observation: All vertices in S must be in any vertex cover of size at most k.
2. If $|S| > k$, stop and return NO. Else, delete all vertices of S from G to get G'.
3. G' has degree at most k, and we need to find a vertex cover of size at most $k - |S|$ in G'.

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).
   Observation: All vertices in S must be in any vertex cover of size at most k.
2. If $|S| > k$, stop and return NO. Else, delete all vertices of S from G to get G'.
3. G' has degree at most k, and we need to find a vertex cover of size at most $k - |S|$ in G'.
4. If G' has more than $(k - |S|)k$ edges, stop and return NO. Delete degree 0 vertices of G'.

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).
   Observation: All vertices in S must be in any vertex cover of size at most k.
2. If $|S| > k$, stop and return NO. Else, delete all vertices of S from G to get G'.
3. G' has degree at most k, and we need to find a vertex cover of size at most $k - |S|$ in G'.
4. If G' has more than $(k - |S|)k$ edges, stop and return NO. Delete degree 0 vertices of G'.
5. G' is the *kernel*. G' has at most k^2 edges and at most $2k^2$ vertices.

A kernel for Vertex Cover

1. Include all vertices with degree more than k in a set S. (S is the proposed solution set).
   Observation: All vertices in S must be in any vertex cover of size at most k.
2. If $|S| > k$, stop and return NO. Else, delete all vertices of S from G to get G'.
3. G' has degree at most k, and we need to find a vertex cover of size at most $k - |S|$ in G'.
4. If G' has more than $(k - |S|)k$ edges, stop and return NO. Delete degree 0 vertices of G'.
5. G' is the *kernel*. G' has at most k^2 edges and at most $2k^2$ vertices.
6. Now we can do branching on G'. Resulting run time is $O(kn + (1.28)^k)$.

More on kernels

1. There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.

More on kernels

1. There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.
2. There is an $O(k^2)$ kernel for undirected feedback vertex set (SODA 2009) – uses Hall's like theorem.

More on kernels

1. There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.
2. There is an $O(k^2)$ kernel for undirected feedback vertex set (SODA 2009) – uses Hall's like theorem.
3. *Famous Open problems:* Polynomial sized ($k^{O(1)}$) kernel for
   - 3.1 Directed feedback vertex set?
   - 3.2 Odd cycle transversal (set of vertices whose removal results in a bipartite graph)?

More on kernels

1. There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.
2. There is an $O(k^2)$ kernel for undirected feedback vertex set (SODA 2009) – uses Hall's like theorem.
3. *Famous Open problems:* Polynomial sized ($k^{O(1)}$) kernel for
   3.1 Directed feedback vertex set?
   3.2 Odd cycle transversal (set of vertices whose removal results in a bipartite graph)?
4. Now, very recently (STOC 08, ICALP 08, 09) there is a machinery (composition) to show NON existence of polynomial sized kernal (unless $PH = \Sigma^3$)

More on kernels

1. There is a $2k$ vertex kernel for vertex cover using Nemhauser-Trotter LP based approximation algorithm for vertex cover.
2. There is an $O(k^2)$ kernel for undirected feedback vertex set (SODA 2009) – uses Hall's like theorem.
3. *Famous Open problems:* Polynomial sized ($k^{O(1)}$) kernel for
   3.1 Directed feedback vertex set?
   3.2 Odd cycle transversal (set of vertices whose removal results in a bipartite graph)?
4. Now, very recently (STOC 08, ICALP 08, 09) there is a machinery (composition) to show NON existence of polynomial sized kernal (unless $PH = \Sigma^3$)
5. k-path, k-connected vertex cover are some problems that don't have polynomial sized kernel under this hypothesis.

Iterated Compression

1. A simple, but powerful technique (for minimization problems)

Iterated Compression

1. A simple, but powerful technique (for minimization problems)
2. 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.

Iterated Compression

1. A simple, but powerful technique (for minimization problems)
2. 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.
3. How do we get the given $k+1$-sized solution? We iterate and compress!

Iterated Compression

1. A simple, but powerful technique (for minimization problems)
2. 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.
3. How do we get the given $k+1$-sized solution? We iterate and compress!
4. Label the vertices $v_1, \ldots v_n$, $\{v_1, ...v_{k+1}\}$ is a $k+1$-sized solution for $G_{k+1} = G[\{v_1, ...v_{k+1}\}]$. Apply the compression step, if this can not be compressed, G_{k+1} has no k sized solution and G also has no k-sized solution.

Iterated Compression

1. A simple, but powerful technique (for minimization problems)
2. 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.
3. How do we get the given $k+1$-sized solution? We iterate and compress!
4. Label the vertices $v_1, \ldots v_n$, $\{v_1, ... v_{k+1}\}$ is a $k+1$-sized solution for $G_{k+1} = G[\{v_1, ... v_{k+1}\}]$. Apply the compression step, if this can not be compressed, G_{k+1} has no k sized solution and G also has no k-sized solution.
5. If G_{k+1} has a k-sized solution S, then $S \cup \{v_{k+2}\}$ is a $(k+1)$-sized solution for G_{k+2} and continue like this.
6. Overall time is $O((n-k) * \textit{time for compression step})$.

Compression step for Feedback vertex set

- *Problem:* Given that G has an FVS S of size $k+1$, check whether it has one of size k.

Compression step for Feedback vertex set

- *Problem:* Given that G has an FVS S of size $k+1$, check whether it has one of size k. The proposed solution intersects S in some subset X such that $0 \leq |X| \leq k$. Hence,

Compression step for Feedback vertex set

- *Problem:* Given that G has an FVS S of size $k+1$, check whether it has one of size k. The proposed solution intersects S in some subset X such that $0 \leq |X| \leq k$. Hence,
- for every subset $X \subseteq S$, $|X| \leq k$, we will check whether there is an FVS of G of size at most k containing all of X and none of $S - X$.

Compression step for Feedback vertex set

- *Problem:* Given that G has an FVS S of size $k+1$, check whether it has one of size k. The proposed solution intersects S in some subset X such that $0 \leq |X| \leq k$. Hence,
- for every subset $X \subseteq S$, $|X| \leq k$, we will check whether there is an FVS of G of size at most k containing all of X and none of $S - X$.
- Since there are at most $2^{k+1} - 1$ such subsets X, the problem is FPT if we show it FPT for a fixed X.

New Problem

- G has an FVS S of size $k+1$.

New Problem

- G has an FVS S of size $k+1$.
- X is a fixed subset of S of size at most k.

New Problem

- G has an FVS S of size $k+1$.
- X is a fixed subset of S of size at most k.
- We look for an FVS of $G - X$ of size at most $k - |X|$ containing no vertex of $S - X$.
- X can be deleted from G.

The Compression problem for FVS

Given $G = (V, E)$, a subset S, which is a FVS of size $k+1$, a subset X of S of size at most k, find a FVS of size at most $k - |X|$ from $V - S$.

The Compression problem for FVS

Given $G = (V, E)$, a subset S, which is a FVS of size $k + 1$, a subset X of S of size at most k, find a FVS of size at most $k - |X|$ from $V - S$.

Observations:

The Compression problem for FVS

Given $G = (V, E)$, a subset S, which is a FVS of size $k + 1$, a subset X of S of size at most k, find a FVS of size at most $k - |X|$ from $V - S$.

Observations:

1. $G[V - S]$ and $G[S - X]$ are forests.

The Compression problem for FVS

Given $G = (V, E)$, a subset S, which is a FVS of size $k+1$, a subset X of S of size at most k, find a FVS of size at most $k - |X|$ from $V - S$.

Observations:

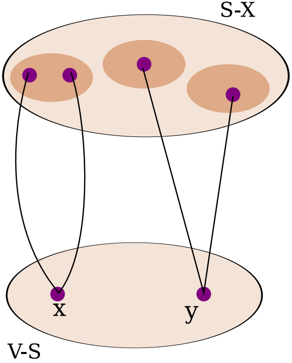
1. $G[V - S]$ and $G[S - X]$ are forests.
2. $G[S - X]$ has at most $k + 1 - |X|$ vertices and hence at most that many components

FPT Algorithm for the New Problem

$G[S - X]$ and $G[V - S]$ are forests; find an FVS of size at most $k - |X|$ from $V - S$ (assume minimum degree 3).

FPT Algorithm for the New Problem

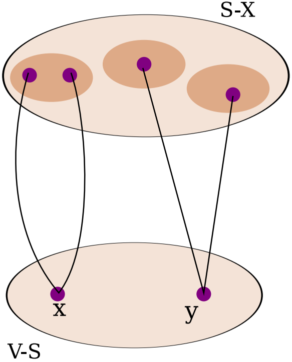
$G[S - X]$ and $G[V - S]$ are forests; find an FVS of size at most $k - |X|$ from $V - S$ (assume minimum degree 3).



The Algorithm:

FPT Algorithm for the New Problem

$G[S - X]$ and $G[V - S]$ are forests; find an FVS of size at most $k - |X|$ from $V - S$ (assume minimum degree 3).

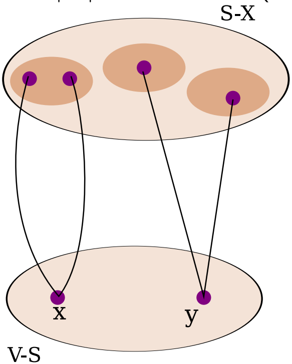


The Algorithm:

1. Let x be a vertex of degree at most 1 in $G[V - S]$. It has at least *two* neighbors in $S - X$.

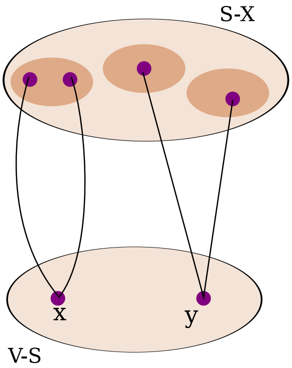
FPT Algorithm for the New Problem

$G[S-X]$ and $G[V-S]$ are forests; find an FVS of size at most $k-|X|$ from $V-S$ (assume minimum degree 3).

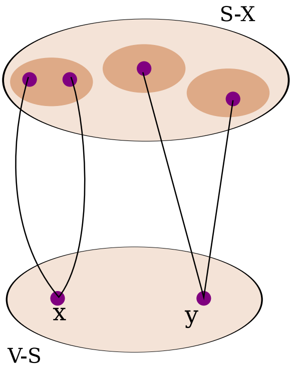


The Algorithm:

1. Let x be a vertex of degree at most 1 in $G[V-S]$. It has at least *two* neighbors in $S-X$.
2. If both neighbors are in the same component of $G[S-X]$, then include x in FVS (forced).



1. Else (let y be a vertex whose neighbors are in at least two components of $G[S-X]$), we branch
   1.1 by picking y in FVS, in which case k drops by 1 in the recursion, or



1. Else (let y be a vertex whose neighbors are in at least two components of $G[S - X]$), we branch
    1.1 by picking y in FVS, in which case k drops by 1 in the recursion, or
    1.2 by not picking y in which case y is added to $S - X$ reducing the number of components of $S - X$ by 1.

Time Complexity

1. In either branch, $k +$ number of components in $G[S - X]$ decreases.

Time Complexity

1. In either branch, $k +$ number of components in $G[S - X]$ decreases.
2. Since $G[S - X]$ had at most k components in the beginning, the depth of the recursion is at most $2k$, and hence the runtime is at most $4^k \text{poly}(n)$.

Time Complexity

1. In either branch, $k +$ number of components in $G[S - X]$ decreases.
2. Since $G[S - X]$ had at most k components in the beginning, the depth of the recursion is at most $2k$, and hence the runtime is at most $4^k \text{poly}(n)$.
3. Overall runtime (with $2^{k+1} - 1$ choices for X) is $4^k 2^k \text{poly}(n)$.
4. With a careful analysis, this turns out to be $O(5^k n^{O(1)})$.
5. This is the best known bound, the open problem is to improve 5^k.

More on Iterated Compression

Several recent results were shown FPT using iterated compression

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

Color coding for finding k-path

- A simple randomized technique (Alon, Yuster, Zwick JACM 95)

Color coding for finding k-path

- A simple 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 simple 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.

Color coding for finding k-path

- A simple 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).

Color coding for finding k-path

- A simple 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.

Color coding for finding k-path

- A simple 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. $O(2^k m)$

Color coding for finding k-path

- A simple 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. If there is a simple path of length k, it will be colorful with probability

Color coding for finding k-path

- A simple 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. If there is a simple path of length k, it will be colorful with probability $k!/k^k$ which is $\Omega(e^{-k})$

Color coding for finding k-path

- A simple 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. If there is a simple path of length k, it will be colorful with probability $k!/k^k$ which is $\Omega(e^{-k})$
  4. Repeat if not found; expected # of repetitions – $O(e^k)$.

Color coding for finding k-path

- A simple 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. If there is a simple path of length k, it will be colorful with probability $k!/k^k$ which is $\Omega(e^{-k})$
  4. Repeat if not found; expected # of repetitions – $O(e^k)$.
  5. 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. Recently applied to get a $2^{O(\sqrt{k}\log k)} + n^{O(1)}$ algorithm for finding Feedback Arc Set in tournaments (ICALP 2009).

Summary of Algorithmic Techniques

1. Graph Minor Theory, MSO, Treewidth machinery (mainly for Classification)
2. Bounded Search Trees
3. Reduction to Kernel
4. Iterated Compression
5. Color Coding

Outline

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

   So B is in FPT implies A is in FPT.

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

   So B is in FPT implies A is in FPT.
2. Canonical complete problems:
   2.1 $W[1]$ – Given a bounded CNF formula, does it have a weight k satisfying assignment?

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

   So B is in FPT implies A is in FPT.
2. Canonical complete problems:
   2.1 $W[1]$ – Given a bounded CNF formula, does it have a weight k satisfying assignment?
   k-Independent Set, k-Clique are $W[1]$-complete.

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

   So B is in FPT implies A is in FPT.
2. Canonical complete problems:
   2.1 $W[1]$ – Given a bounded CNF formula, does it have a weight k satisfying assignment?
   k-Independent Set, k-Clique are $W[1]$-complete.
   Also short TM acceptance (does the given NDTM accept a string in k steps).

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

   So B is in FPT implies A is in FPT.
2. Canonical complete problems:
   2.1 $W[1]$ – Given a bounded CNF formula, does it have a weight k satisfying assignment?
   k-Independent Set, k-Clique are $W[1]$-complete.
   Also short TM acceptance (does the given NDTM accept a string in k steps).
   2.2 $W[2]$ – Given an unbounded CNF formula, does it have a weight k satisfying assignment?

The hardness theory and reductions

1. Parameterized reductions – An algorithm that reduces (x, k) of problem A to equivalent (x', k') of problem B
   - time allowed is $f(k)|x|^c$ time,
   - but k' must be a function of k.

   So B is in FPT implies A is in FPT.
2. Canonical complete problems:
   2.1 $W[1]$ – Given a bounded CNF formula, does it have a weight k satisfying assignment?
   k-Independent Set, k-Clique are $W[1]$-complete.
   Also short TM acceptance (does the given NDTM accept a string in k steps).
   2.2 $W[2]$ – Given an unbounded CNF formula, does it have a weight k satisfying assignment?
   k-Dominating Set, k-hitting set are $W[2]$ complete.

Outline

Approximation and FPT

1. For all maximization problems in MAXSNP, their corresponding decision question is in FPT (with the solution size as the parameter)
2. There are easy to approximate problems whose decision versions are W-hard (rectangle stabbing) and
3. There are FPT problems (k-path, odd cycle traversal) whose optimization versions are hard to approximate.
4. EPTAS implies the corresponding decision version is FPT.
5. Last word not out yet!

Outline

Recent Trends

Recent Trends

1. Lower and upper bounds for kernels

Recent Trends

1. Lower and upper bounds for kernels
2. Subexponential Algorithms in restricted classes of graphs (bidimensionality theory and beyond)

Recent Trends

1. Lower and upper bounds for kernels
2. Subexponential Algorithms in restricted classes of graphs (bidimensionality theory and beyond)
3. More width parameters (treewidth, pathwidth, cliquewidth, boolean width, ...)

Recent Trends

1. Lower and upper bounds for kernels
2. Subexponential Algorithms in restricted classes of graphs (bidimensionality theory and beyond)
3. More width parameters (treewidth, pathwidth, cliquewidth, boolean width, ...)
4. Inclusion-Exclusion applications (finding and counting structures)

Recent Trends

1. Lower and upper bounds for kernels
2. Subexponential Algorithms in restricted classes of graphs (bidimensionality theory and beyond)
3. More width parameters (treewidth, pathwidth, cliquewidth, boolean width, ...)
4. Inclusion-Exclusion applications (finding and counting structures)
5. New techniques, new problems, new applications (Geometry, AI, ...?)

Recent Trends

1. Lower and upper bounds for kernels
2. Subexponential Algorithms in restricted classes of graphs (bidimensionality theory and beyond)
3. More width parameters (treewidth, pathwidth, cliquewidth, boolean width, ...)
4. Inclusion-Exclusion applications (finding and counting structures)
5. New techniques, new problems, new applications (Geometry, AI, ...?)
6. FPT Approximation for W-hard problems

Outline

Conclusions

1. Reasonably young, at the same time, has reasonably rich theory
2. Well-developed techniques – some simple, some use heavy machinery
3. Lots of open problems, combinatorial results
4. Practical applications in computational biology, optimization

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 IWPEC, and other conferences

Thank You!