Algorithms and Complexity

Fixed Parameter Algorithms

Marc Hellmuth

University of Stockholm

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Three general desired features of an algorithm:

1. “Solve” (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Three general desired features of an algorithm:

1. "Solve" (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

Unless $P = NP$, an algorithm can have two of these three features, but not all three.

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Three general desired features of an algorithm:

1. "Solve" (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

Unless $P = NP$, an algorithm can have two of these three features, but not all three.

Feature 2+3: polynomial-time exact algorithm (in P)

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Three general desired features of an algorithm:

1. "Solve" (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

Unless $P = NP$, an algorithm can have two of these three features, but not all three.

Feature 2+3: polynomial-time exact algorithm (in P)

Feature 1+2: e.g. approximation algorithms

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Three general desired features of an algorithm:

1. "Solve" (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

Unless $P = NP$, an algorithm can have two of these three features, but not all three.

Feature 2+3: polynomial-time exact algorithm (in P)

Feature 1+2: e.g. approximation algorithms

Feature 1+3: Fixed-parameter algorithms

Fixed Parameter Algorithms

Fixed Parameter Algorithms are an alternative way to deal with NP-hard problems instead of approximation algorithms.

Three general desired features of an algorithm:

1. “Solve” (NP-)hard problems
2. Run in polynomial time (fast)
3. Get exact solutions

Unless $P = NP$, an algorithm can have two of these three features, but not all three.

Feature 2+3: polynomial-time exact algorithm (in P)

Feature 1+2: e.g. approximation algorithms

Feature 1+3: Fixed-parameter algorithms

Idea: Aim an exact algorithm but isolate exponential runtime to a specific parameter. When the value of this parameter is small, the algorithm gets fast.

Fixed Parameter Algorithms

A parameterized problem (Π, k) is a pair consisting of

- decision problem Π and
- a parameter k, i.e., a map k that assigns to each instance $I \in \Pi$ a non-negative integer $k(I)$.
  (often write $k_I := k(I)$)

Fixed Parameter Algorithms

A parameterized problem (Π, k) is a pair consisting of

- decision problem Π and
- a parameter k, i.e., a map k that assigns to each instance $I \in \Pi$ a non-negative integer $k(I)$.

  (often write $k_I := k(I)$)

Example:

Decision problem Π.

input: (G, K)

question: Is there a path of length $\leq K$ between x and y in G ?

parameter could be $k_G = K$ or $k_G =$ maximum degree in G

Fixed Parameter Algorithms

A parameterized problem (Π, k) is a pair consisting of

- decision problem Π and
- a parameter k, i.e., a map k that assigns to each instance $I \in \Pi$ a non-negative integer $k(I)$.
  (often write $k_I := k(I)$)

Example:

Decision problem Π.
input: (G, K)
question: Is there a path of length $\leq K$ between x and y in G ?

parameter could be $k_G = K$ or $k_G =$ maximum degree in G

Many ”natural” parameter exist, but we are interested in particular one!

Idea: Specify a parameter that isolates the exponential runtime of an exact algorithm for Π. When the value of this parameter is small, the algorithm gets fast.

Fixed Parameter Algorithms

Brute-force Vertex Cover : $O(|I|^{k_I})$ (bad!) (WHITEBOARD)

Fixed Parameter Algorithms

Brute-force Vertex Cover : $O(|I|^{k_I})$ (bad!) (WHITEBOARD)

A parameterized problem (Π, k) is fixed-parameter tractable (FPT) if there is an algorithm that, for all $I \in \Pi$, solves/decides I (yes or no) in time $\leq f(k_I) \cdot |I|^{O(1)}$,
where $f : \mathbb{N} \to \mathbb{N}$ (non negative) and $O(1)$ degree in $|I|^{O(1)}$ is independent of k_I and n.

The class FPT consists of all fixed-parameter tractable problems (Π, k).

Fixed Parameter Algorithms

Brute-force Vertex Cover : $O(|I|^{k_I})$ (bad!) (WHITEBOARD)

A parameterized problem (Π, k) is fixed-parameter tractable (FPT) if there is an algorithm that, for all $I \in \Pi$, solves/decides I (yes or no) in time $\leq f(k_I) \cdot |I|^{O(1)}$,
where $f : \mathbb{N} \to \mathbb{N}$ (non negative) and $O(1)$ degree in $|I|^{O(1)}$ is independent of k_I and n.

The class FPT consists of all fixed-parameter tractable problems (Π, k).

Fixed Parameter Algorithms

Brute-force Vertex Cover : $O(|I|^{k_I})$ (bad!) (WHITEBOARD)

A parameterized problem (Π, k) is fixed-parameter tractable (FPT) if there is an algorithm that, for all $I \in \Pi$, solves/decides I (yes or no) in time $\leq f(k_I) \cdot |I|^{O(1)}$,
where $f : \mathbb{N} \to \mathbb{N}$ (non negative) and $O(1)$ degree in $|I|^{O(1)}$ is independent of k_I and n.

The class FPT consists of all fixed-parameter tractable problems (Π, k).

An FPT-algorithm for Vertex Cover : $O(2^{k_I} \cdot |I|)$ (good) (WHITEBOARD)

Fixed Parameter Algorithms

Brute-force Vertex Cover : $O(|I|^{k_I})$ (bad!) (WHITEBOARD)

A parameterized problem (Π, k) is fixed-parameter tractable (FPT) if there is an algorithm that, for all $I \in \Pi$, solves/decides I (yes or no) in time $\leq f(k_I) \cdot |I|^{O(1)}$, where $f : \mathbb{N} \to \mathbb{N}$ (non negative) and $O(1)$ degree in $|I|^{O(1)}$ is independent of k_I and n.

The class FPT consists of all fixed-parameter tractable problems (Π, k).

An FPT-algorithm for Vertex Cover : $O(2^{k_I} \cdot |I|)$ (good) (WHITEBOARD)

Question: why not even aiming at an $f(k_I) + |I|^{O(1)}$ time algorithm?

Theorem. $\exists\, f(k_I) \cdot |I|^c$ algorithm $\iff$ $\exists\, \tilde{f}(k_I) + |I|^{\tilde{c}}$ algorithm (WHITEBOARD)

Fixed Parameter Algorithms

To show that a parameterized problem is FPT there are two general techniques.

General Techniques:

- Bounded search-tree

  General Idea: ”exhaustive” search (i.e., full enumeration of all possible solutions) is conducted in a suitable search tree with limited depth.

  have seen vertex-cover example

- Kernelization

  General Idea: reduce instance to a (possibly still NP-hard difficult) problem kernel by applying various rules.

  let’s focus on this now

Kernelization

General Idea: reduce instance to a (possibly still NP-hard difficult) problem kernel by applying various rules.

Kernelization is transformation of $(I, k_I) \in (\Pi, k)$ to an instance $(I', k_{I'}) \in (\Pi, k)$ such that

- I is yes-instance of $\Pi \iff I'$ is yes-instance of Π
- $|I'| \leq \tilde{f}(k_I)$ for some $\tilde{f}: \mathbb{N} \to \mathbb{N}$, i.e., size of instance I' only depends on parameter k_I
- $k_{I'} \leq k_I$, i.e., parameter $k(I')$ does not increase
- Transformation can be achieved in polynomial time

Theorem. A problem (Π, k) is FPT $\iff$ there exist a Kernelization of (Π, k) (WHITEBOARD)

Kernelization for Vertex Cover : (WHITEBOARD)