Parameterized Complexity

Computability, Algorithms & more

Elli Anastasiadi

May 22, 2020

Háskólinn í Reykjavík

Corelab NTUA

Contents

Introduction

Facing Untractability via Parameters

When he hit the wall of NP-completeness we try other methods

- Probabilistic Algorithms
- Approximations

Facing Untractability via Parameters

When he hit the wall of NP-completeness we try other methods

- Probabilistic Algorithms
- Approximations

Through Parameterized algorithms we avoid some flaws of the above by searching solutions for only part or the universe of the instances.

Facing Untractability via Parameters

When he hit the wall of NP-completeness we try other methods

- Probabilistic Algorithms
- Approximations

Through Parameterized algorithms we avoid some flaws of the above by searching solutions for only part or the universe of the instances.

Parameters
We correlate a problem with a parameter (a computable function) and design algorithms on the notion that our instances have the given parameter bounded and in general terms small.

Examples

- In Optimisation problems one of the most common parameters is the **size** of the solution (called natural parameterization)

Examples

- In Optimisation problems one of the most common parameters is the **size** of the solution (called natural parameterization)
- On Constraint problems (such as SAT) we often parameterize by the **number of constraints**

Examples

- In Optimisation problems one of the most common parameters is the **size** of the solution (called natural parameterization)
- On Constraint problems (such as SAT) we often parameterize by the **number of constraints**
- For properties of graphs we often use **structural** parameters such as max degree , colour number and other easy or hard to compute properties (tree-width, clique-width, branch-width)

Some guys walk into a bar

In a town the Doorman of a bar must choose who he lets in so that there will be no feuds between them. We represent the people by vertices and the feuds by edges between them.

Some guys walk into a bar

In a town the Doorman of a bar must choose who he lets in so that there will be no feuds between them. We represent the people by vertices and the feuds by edges between them.

Vertex Cover
Given a graph G=(V,E) find the min number of vertices to delete so there will be no edges left in the graph.

Some guys walk into a bar

In a town the Doorman of a bar must choose who he lets in so that there will be no feuds between them. We represent the people by vertices and the feuds by edges between them.

Vertex Cover
Given a graph G=(V,E) find the min number of vertices to delete so there will be no edges left in the graph.

Official Parameterized Version
Input: A Graph G=(V,E) Parameter: A Positive Integer k Output: Does G have a VC of size k?

Since this is an optimisation problem as we already mentioned we usually use as a parameter the size of the solution.

Solving Vertex Cover

Main Idea
The running time or an algorithms usually explodes when there is branching affected by the size. Make the branching bound by the parameter and we will have the requested time bound. .

Lets try an algorithm.

1. Begin with the root node labelled by zero. (Represents the an empty VC containing none of the V nodes)

Solving Vertex Cover

Main Idea
The running time or an algorithms usually explodes when there is branching affected by the size. Make the branching bound by the parameter and we will have the requested time bound. .

Lets try an algorithm.

1. Begin with the root node labelled by zero. (Represents the an empty VC containing none of the V nodes)
2. Pick one edge (u,v) of the current graph and branch two children one containing u and one containing v (and label accordingly).

Main Idea
The running time or an algorithms usually explodes when there is branching affected by the size. Make the branching bound by the parameter and we will have the requested time bound. .

Lets try an algorithm.

1. Begin with the root node labelled by zero. (Represents the an empty VC containing none of the V nodes)
2. Pick one edge (u,v) of the current graph and branch two children one containing u and one containing v (and label accordingly).
3. update the graph by each time deleting the node's label vertices.

Solving Vertex Cover

Main Idea
The running time or an algorithms usually explodes when there is branching affected by the size. Make the branching bound by the parameter and we will have the requested time bound. .

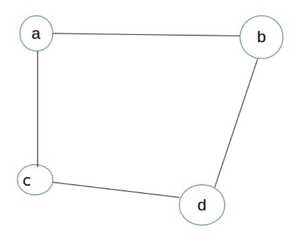
Lets try an algorithm.

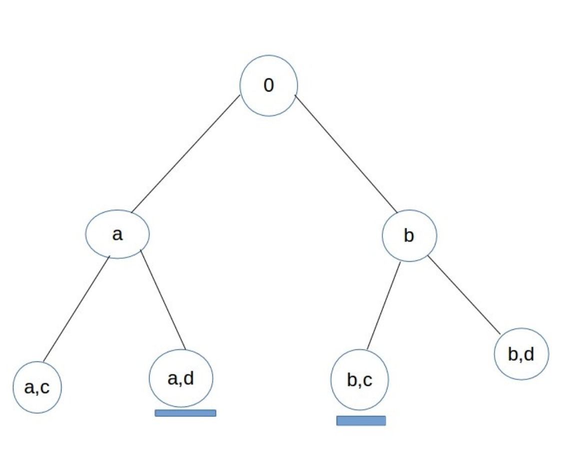
1. Begin with the root node labelled by zero. (Represents the an empty VC containing none of the V nodes)
2. Pick one edge (u,v) of the current graph and branch two children one containing u and one containing v (and label accordingly).
3. update the graph by each time deleting the node's label vertices.
4. repeat k times

Example

We will check a Graph for a VC of size 2

G:





Cont.

What me manage this way :

- Resulting tree is of depth k.
- Each level i of the resulting tree T has nodes with exactly i vertices in the label.
- If there is a leaf that by deleting its label's vertices from G there is no edge left in G then this set of vertices is a VC of size k.

Is the above procedure correct? yes! For each edge we have to delete at least one end at some point. Since we explore both options if there is a VC of size k this algorithm will find it .

Formal Definitions

Tractability

Parameterized Problem
A *parameterization* of Σ^* is a *recursive* function $k : \Sigma^* \to \mathbb{N}$. A *parameterized problem* is a tuple (L, k), where $L \subseteq \Sigma^*$ and k is a parameterization of Σ^*.

Parameterized Problem
A *parameterization* of Σ^* is a *recursive* function $k : \Sigma^* \to \mathbb{N}$. A *parameterized problem* is a tuple (L, k), where $L \subseteq \Sigma^*$ and k is a parameterization of Σ^*.

The Class FPT
The class of parameterized problems that can be solved in time

$$O(f(k) * n^c)$$

, where f(k) is computable.

Tractability

Parameterized Problem
A *parameterization* of Σ^* is a *recursive* function $k : \Sigma^* \to \mathbb{N}$. A *parameterized problem* is a tuple (L, k), where $L \subseteq \Sigma^*$ and k is a parameterization of Σ^*.

The Class FPT
The class of parameterized problems that can be solved in time

$$O(f(k) * n^c)$$

, where f(k) is computable.

As always the classification of problems in classes refers to the best known algorithm or reduction for a parameterized problem

Picking your weapon

One can design an algorithm that runs in the above time if he chooses the parameter cunningly.

- We could try parameterizing a problem by the parameter n-1 where n is input size.

Picking your weapon

One can design an algorithm that runs in the above time if he chooses the parameter cunningly.

- We could try parameterizing a problem by the parameter n-1 where n is input size.
- This is allowed!!

Picking your weapon

One can design an algorithm that runs in the above time if he chooses the parameter cunningly.

- We could try parameterizing a problem by the parameter n-1 where n is input size.
- This is allowed!! Not very useful though. why?

Picking your weapon

One can design an algorithm that runs in the above time if he chooses the parameter cunningly.

- We could try parameterizing a problem by the parameter n-1 where n is input size.
- This is allowed!! Not very useful though. why?

1. The parameter will be considered constant and small - We have to choose it in a way that is realistic .

Picking your weapon

One can design an algorithm that runs in the above time if he chooses the parameter cunningly.

- We could try parameterizing a problem by the parameter n-1 where n is input size.
- This is allowed!! Not very useful though. why?

1. The parameter will be considered constant and small - We have to choose it in a way that is realistic .
2. The instances that have the parameter satisfying the above should be as many as possible

Definition
Given (L, k), (L', k') parameterized problems. We say that (L, k) reduces to (L', k') through an FPT-reduction (note . $L \leq_{FPT} L'$) iff there exists algorithm R such that:

1. $\forall\ x \in \Sigma^*$, $x \in L \Leftrightarrow R(x) \in L'$
2. R is computable by and FPT-algorithm.
3. $k' = g(k)$, where $g : \mathbb{N} \to \mathbb{N}$ computable function.

If $A \leq_{FPT} B$ and $B \leq_{FPT} A$, then we say that are A, B are FPT-equivalent (note. $A \equiv_{FPT} B$).

Definition
Given (L, k), (L', k') parameterized problems. We say that (L, k) reduces to (L', k') through an FPT-reduction (note . $L \leq_{FPT} L'$) iff there exists algorithm R such that:

1. $\forall\ x \in \Sigma^*$, $x \in L \Leftrightarrow R(x) \in L'$
2. R is computable by and FPT-algorithm.
3. $k' = g(k)$, where $g : \mathbb{N} \to \mathbb{N}$ computable function.

If $A \leq_{FPT} B$ and $B \leq_{FPT} A$, then we say that are A, B are FPT-equivalent (note. $A \equiv_{FPT} B$).

Example
Clique to Independent Set ,CNF sat to Waited Integer Programming are $\leq_{FPT}$.

Definition
Given (L,k), (L',k') parameterized problems. We say that (L,k) reduces to (L',k') through an FPT-reduction (note . $L \leq_{FPT} L'$) iff there exists algorithm R such that:

1. $\forall\ x \in \Sigma^*,\ x \in L \Leftrightarrow R(x) \in L'$
2. R is computable by and FPT-algorithm.
3. $k' = g(k)$, where $g : \mathbb{N} \to \mathbb{N}$ computable function.

If $A \leq_{FPT} B$ and $B \leq_{FPT} A$, then we say that are A, B are FPT-equivalent (note. $A \equiv_{FPT} B$).

Example
Clique to Independent Set ,CNF sat to Waited Integer Programming are $\leq_{FPT}$.
Vertex Cover to Clique is not. (why?)

Techniques based on graph structure

Other Metrics

There are many ways to parameterize a problem.

- you can parameterize by rank of a Matrix
- by the eccentricity of a vertex
- by the density of a graph

BUT! : As we mentioned you have to be sure that by assuming the parameter bound and small you are not ignoring important or common instances of a problem.

Other Metrics

There are many ways to parameterize a problem.

- you can parameterize by rank of a Matrix
- by the eccentricity of a vertex
- by the density of a graph

BUT! : As we mentioned you have to be sure that by assuming the parameter bound and small you are not ignoring important or common instances of a problem.
What have we came up with? Treewidth! (and many other graph metrics).

Other Metrics

There are many ways to parameterize a problem.

- you can parameterize by rank of a Matrix
- by the eccentricity of a vertex
- by the density of a graph

BUT! : As we mentioned you have to be sure that by assuming the parameter bound and small you are not ignoring important or common instances of a problem.

What have we came up with? Treewidth! (and many other graph metrics).

Treewidth is a graph metric we use to define in a way how far a graph is from a tree.

Other Metrics

There are many ways to parameterize a problem.

- you can parameterize by rank of a Matrix
- by the eccentricity of a vertex
- by the density of a graph

BUT! : As we mentioned you have to be sure that by assuming the parameter bound and small you are not ignoring important or common instances of a problem.
What have we came up with? Treewidth! (and many other graph metrics).
Treewidth is a graph metric we use to define in a way how far a graph is from a tree.
Why?

Other Metrics

There are many ways to parameterize a problem.

- you can parameterize by rank of a Matrix
- by the eccentricity of a vertex
- by the density of a graph

BUT! : As we mentioned you have to be sure that by assuming the parameter bound and small you are not ignoring important or common instances of a problem.
What have we came up with? Treewidth! (and many other graph metrics).
Treewidth is a graph metric we use to define in a way how far a graph is from a tree.
Why? Because trees are nice.

Definitions

1. A Tree decomposition of a graph $G = (V, E)$ is a tree T together with a collection of subsets T_x (called bags) of V labeled with the vertices x of T such that $\cup T_x = V$ and the following hold

Definitions

1. A Tree decomposition of a graph $G = (V, E)$ is a tree T together with a collection of subsets T_x (called bags) of V labeled with the vertices x of T such that $\cup T_x = V$ and the following hold
    - For every edge (u, v) of G there is a some x such that $(u, v) \in T_x$

Definitions

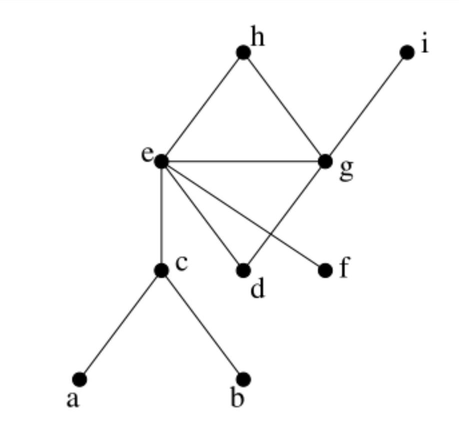
1. A Tree decomposition of a graph $G = (V, E)$ is a tree T together with a collection of subsets T_x (called bags) of V labeled with the vertices x of T such that $\cup T_x = V$ and the following hold
    - For every edge (u, v) of G there is a some x such that $(u, v) \in T_x$
    - If y is a vertex of on the unique path in T from x to z then $T_x \cap T_z \subseteq T_y$.

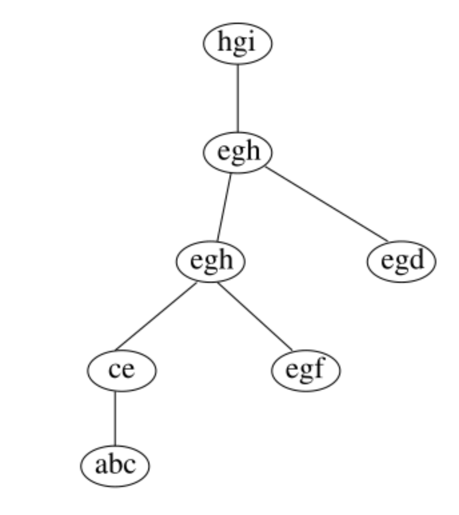
Definitions

1. A Tree decomposition of a graph $G = (V, E)$ is a tree T together with a collection of subsets T_x (called bags) of V labeled with the vertices x of T such that $\cup T_x = V$ and the following hold
   - For every edge (u, v) of G there is a some x such that $(u, v) \in T_x$
   - If y is a vertex of on the unique path in T from x to z then $T_x \cap T_z \subseteq T_y$.
2. The width of a tree decomposition is the maximum value of $|T_x| - 1$ over all the vertices of the tree T of the decomposition.

Definitions

1. A Tree decomposition of a graph $G = (V, E)$ is a tree T together with a collection of subsets T_x (called bags) of V labeled with the vertices x of T such that $\cup T_x = V$ and the following hold
   - For every edge (u, v) of G there is a some x such that $(u, v) \in T_x$
   - If y is a vertex of on the unique path in T from x to z then $T_x \cap T_z \subseteq T_y$.
2. The width of a tree decomposition is the maximum value of $|T_x| - 1$ over all the vertices of the tree T of the decomposition.
3. The treewidth of Graph G is the minimum treewidth of all thee decompositions of G.


h
i
e
g
c
d
f
a
b


hgi
egh
egh
egd
ce
egf
abc

Independent Set

Given a Graph G find the maximum Set L such that if $u, v \in L$ then $(u, v) \notin E$

Independent Set

Given a Graph G find the maximum Set L such that if $u, v \in L$ then $(u, v) \notin E$

This Problem is NP-hard

BUT!: Many of the real-world problems that require us to check this property have bounded treewidth! So:

The Algorithm

1. Given a Graph and a tree Decomposition of tw k. (we will use the one given in the previous example)
2. For each node of T we construct a vector with 2^k positions as follows

Ø	a	b	c	ab	ac	bc	abc
0	1	1	1	2	–	–	–

We store in each position of the vector the size of the larger Independent set this far. That is the size of the set corresponding to the vectors bit plus the size of the previously larger Independent set for the vectors already filled.

Cont.

Of course we are careful if the current bit of the vector has common vertices with the previous max independent set . But we only have to make this check for adjacent nodes of T . Continuing by adding up independent sets for empty leaf nodes we get the Max independent set.
We Can pause here and try it for the above tree decomposition.
The proof of this algorithm can be found in the literature given later.

Hierarchy

An intractable problem

How do we find an intractable problem? We need a model!

An intractable problem

How do we find an intractable problem? We need a model!

- Classical complexity $\rightarrow$ Turing machines
- Probabilistic algorithms $\rightarrow$ Turing Machines that might be wrong (or not fast)
- Approximation algorithms $\rightarrow$ Turing machines with error.

An intractable problem

How do we find an intractable problem? We need a model!

- Classical complexity $\rightarrow$ Turing machines
- Probabilistic algorithms $\rightarrow$ Turing Machines that might be wrong (or not fast)
- Approximation algorithms $\rightarrow$ Turing machines with error.

So.... parameterized Turing machines?

An intractable problem

How do we find an intractable problem? We need a model!

- Classical complexity $\to$ Turing machines
- Probabilistic algorithms $\to$ Turing Machines that might be wrong (or not fast)
- Approximation algorithms $\to$ Turing machines with error.

So.... parameterized Turing machines?

SHORT TURING MACHINE ACCEPTANCE

Input: A nondeterministic Turing machine M and a string x

Parameter: A positive integer k.

Question: Does M have a computation path accepting x in $\leq$ k steps

An intractable problem

How do we find an intractable problem? We need a model!

- Classical complexity $\rightarrow$ Turing machines
- Probabilistic algorithms $\rightarrow$ Turing Machines that might be wrong (or not fast)
- Approximation algorithms $\rightarrow$ Turing machines with error.

So.... parameterized Turing machines?

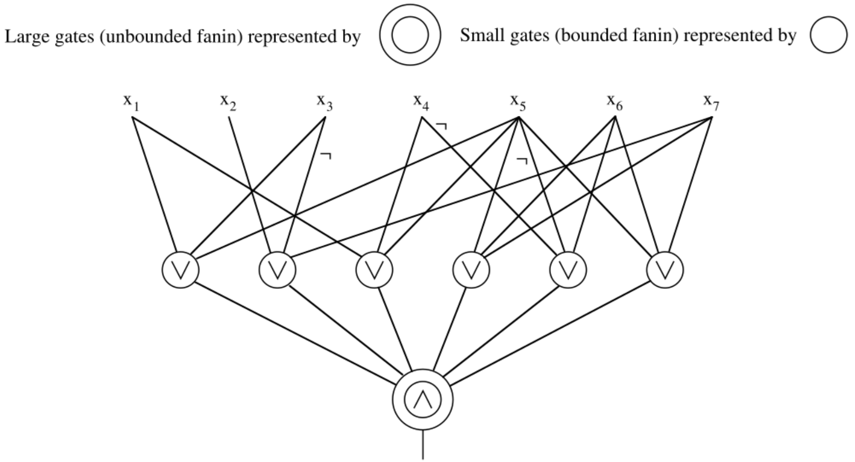
SHORT TURING MACHINE ACCEPTANCE

Input: A nondeterministic Turing machine M and a string x
Parameter: A positive integer k.
Question: Does M have a computation path accepting x in $\leq$ k steps

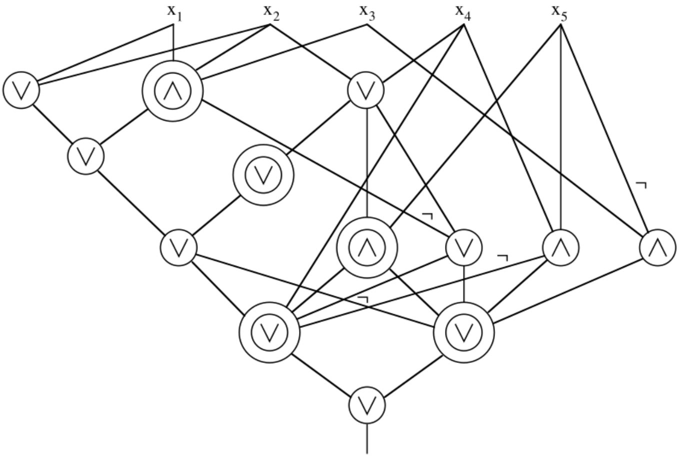
Can we build a hierarchy on this problem? nope.

- We need a naturally parameterized problem (the chosen parameter for Turing machine acceptance could be anything)
- Clasical complexity: Cooks theorem (Turing machine $\sim$ SAT).
- We will try to do the same.

Weft
Let C be a decision circuit. The weft of C is defined to be the maximum number of large gates on any path from the input variables to the output line. (A gate is called large if its fan-in exceeds some preassigned bound.)



A 3CNF Formula is a large and of small or's.



A Weft 2 Depth 5 Decision Circuit.

Intractability

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY (*WCS(t, h)*)

Input:A weft t depth h decision circuit C

Parameter: A positive integer k

Question: Does C have a weight k satisfying assignment?

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY (*WCS(t, h)*)
Input:A weft t depth h decision circuit C
Parameter: A positive integer k
Question: Does C have a weight k satisfying assignment?

Remarks

- The i^{th} level of the hierarchy corresponds to problems reducible to weft i circuits
- Circuit depth is essentially irrelevant (but bounded).
- A large gate is considered one that is fanin is more than $f(k)$

Analog of Cook's Theorem

The following are complete for $W[1]$:

1. WEIGHTED n-SATISFIABILITY for any fixed n ≥ 2
2. SHORT TURING MACHINE ACCEPTANCE

Analog of Cook's Theorem

The following are complete for $W[1]$:

1. WEIGHTED n-SATISFIABILITY for any fixed n ≥ 2
2. SHORT TURING MACHINE ACCEPTANCE

Proof?

Analog of Cook's Theorem

The following are complete for $W[1]$:

1. WEIGHTED n-SATISFIABILITY for any fixed n ≥ 2
2. SHORT TURING MACHINE ACCEPTANCE

Proof? NOPE $\rightarrow$ see chapter 21 of bibliography (1)

Optimisation , Approximation & Connections to FPT

So what have we learned?

Optimisation

So what have we learned?

- We have found more efficient ways to solve decision problems using parameters.

Optimisation

So what have we learned?

- We have found more efficient ways to solve decision problems using parameters.
- The problems we tried this far were all NP-optimisation ones.

Optimisation

So what have we learned?

- We have found more efficient ways to solve decision problems using parameters.
- The problems we tried this far were all NP-optimisation ones.

What is the correlation? Do the algorithms we described remain efficient?

Optimisation

So what have we learned?

- We have found more efficient ways to solve decision problems using parameters.
- The problems we tried this far were all NP-optimisation ones.

What is the correlation? Do the algorithms we described remain efficient?
We will try to formulate this

Optimisation

So what have we learned?

- We have found more efficient ways to solve decision problems using parameters.
- The problems we tried this far were all NP-optimisation ones.

What is the correlation? Do the algorithms we described remain efficient?
We will try to formulate this

DECISION PROBLEM ASSOCIATED WITH AN NP OPTIMIZATION PROBLEM $Q = (I_Q, S_Q, f_Q, opt_Q)$.

Input: $x \in I_Q$.
Parameter: A positive integer k.
Question: Does $R(opt_Q(x), k)$ hold?

Cont

Thanks to Parameterized Complexity Theory we have the following

Theorem. Cai and Chen
Iff you can check if the decision version of an NP optimisation problem in FPT time then you can find the optimal in FPT time.

Cont

Thanks to Parameterized Complexity Theory we have the following

Theorem. Cai and Chen
Iff you can check if the decision version of an NP optimisation problem in FPT time then you can find the optimal in FPT time.

Can you think of a proof?

Cont

Thanks to Parameterized Complexity Theory we have the following

Theorem. Cai and Chen
Iff you can check if the decision version of an NP optimisation problem in FPT time then you can find the optimal in FPT time.

Can you think of a proof?

This is not the actual formulation of the theorem

FPT and Appoximation

1. Pick your favourite optimisation Problem

FPT and Appoximation

1. Pick your favourite optimisation Problem
2. Make it into a Decision one and then parameterize it by the size of the solution

FPT and Appoximation

1. Pick your favourite optimisation Problem
2. Make it into a Decision one and then parameterize it by the size of the solution
3. Is it FTP?

FPT and Appoximation

1. Pick your favourite optimisation Problem
2. Make it into a Decision one and then parameterize it by the size of the solution
3. Is it FTP?

Theorem : Bazgan , Cai and Chen
If a NP-optimisation Problem has a fully Polynomial time approximation scheme then it is FPT

FPT and Appoximation

1. Pick your favourite optimisation Problem
2. Make it into a Decision one and then parameterize it by the size of the solution
3. Is it FTP?

Theorem : Bazgan , Cai and Chen
If a NP-optimisation Problem has a fully Polynomial time approximation scheme then it is FPT

And we are going to prove this

Proof

1. W.L.G Say the problem is a maximisation one

Proof

1. W.L.G Say the problem is a maximisation one
2. Since it has a fully PTAS there is an algorithm A that runs in time $O(p((1/e) * |x|))$ and approximates it by an error of e .

Proof

1. W.L.G Say the problem is a maximisation one
2. Since it has a fully PTAS there is an algorithm A that runs in time $O(p((1/e) * |x|))$ and approximates it by an error of e .
3. We only need to prove that the decision version is FPT

Proof

1. W.L.G Say the problem is a maximisation one
2. Since it has a fully PTAS there is an algorithm A that runs in time $O(p((1/e) * |x|))$ and approximates it by an error of e .
3. We only need to prove that the decision version is FPT
4. For an instance $< x, k >$ run A for $< x, 1/2k >$

Proof

1. W.L.G Say the problem is a maximisation one
2. Since it has a fully PTAS there is an algorithm A that runs in time $O(p((1/e) * |x|))$ and approximates it by an error of e .
3. We only need to prove that the decision version is FPT
4. For an instance $< x, k >$ run A for $< x, 1/2k >$

   What is the running time of this? Find e with respect to k

Proof

1. W.L.G Say the problem is a maximisation one
2. Since it has a fully PTAS there is an algorithm A that runs in time $O(p((1/e) * |x|))$ and approximates it by an error of e .
3. We only need to prove that the decision version is FPT
4. For an instance $< x, k >$ run A for $< x, 1/2k >$

   What is the running time of this? Find e with respect to k

5. • if $k < f(x)$ then $k < opt(x)$ since this is a maximisation problem

Proof

1. W.L.G Say the problem is a maximisation one
2. Since it has a fully PTAS there is an algorithm A that runs in time $O(p((1/e) * |x|))$ and approximates it by an error of e .
3. We only need to prove that the decision version is FPT
4. For an instance $< x, k >$ run A for $< x, 1/2k >$

   What is the running time of this? Find e with respect to k

5. - if $k < f(x)$ then $k < opt(x)$ since this is a maximisation problem
   - if $k > f(x)$ then $k - 1 \geq f(x)$, but $e < 1/k \Rightarrow k > opt(x)$

Proof Cont.

- Therefore $k < f(x)$ iff $k < opt(x)$

Proof Cont.

- Therefore $k < f(x)$ iff $k < opt(x)$
- A runs in time $O(p((2k) * |x|))$

Proof Cont.

- Therefore $k < f(x)$ iff $k < opt(x)$
- A runs in time $O(p((2k) * |x|))$
- Therefore The Problem is FPT

Proof Cont.

- Therefore $k < f(x)$ iff $k < opt(x)$
- A runs in time $O(p((2k) * |x|))$
- Therefore The Problem is FPT

Unfortunately a converse does not exist.

Proof Cont.

- Therefore $k < f(x)$ iff $k < opt(x)$
- A runs in time $O(p((2k) * |x|))$
- Therefore The Problem is FPT

Unfortunately a converse does not exist.
Given a FPT algorithm we cannot guarantee the existence of a fully polynomial time approximation Scheme .

Proof Cont.

- Therefore $k < f(x)$ iff $k < opt(x)$
- A runs in time $O(p((2k) * |x|))$
- Therefore The Problem is FPT

Unfortunately a converse does not exist.
Given a FPT algorithm we cannot guarantee the existence of a fully polynomial time approximation Scheme .
However this allows us to establish that many problems are FPT without effort. For instance:

- Bounded Knapsack
- Planar Independent Set
- Linear Extension Count

Are all FPT.

Approximation

- Say though that a NP-problem is Fixed parameter Intractable.

Approximation

- Say though that a NP-problem is Fixed parameter Intractable.
- So what do we do?

Approximation

- Say though that a NP-problem is Fixed parameter Intractable.
- So what do we do?
- Can we approximate?

Approximation

- Say though that a NP-problem is Fixed parameter Intractable.
- So what do we do?
- Can we approximate? Pause for suspense..

Approximation

- Say though that a NP-problem is Fixed parameter Intractable.
- So what do we do?
- Can we approximate? Pause for suspense..

Of course not.

At least not efficiently

Theorem: Bazgan , Cai and Chen
If a NP-Optimisation Problem is Fixed Parameter Intractable then it has no fully polynomial time approximation Scheme.

At least not efficiently

Theorem: Bazgan , Cai and Chen
If a NP-Optimisation Problem is Fixed Parameter Intractable then it has no fully polynomial time approximation Scheme.
To be more specific under the hypothesis $FPT \neq W[1]$ for parameterized problems There is not fully PTAS for any W[1] problems.

At least not efficiently

Theorem: Bazgan , Cai and Chen
If a NP-Optimisation Problem is Fixed Parameter Intractable then it has no fully polynomial time approximation Scheme.
To be more specific under the hypothesis $FPT \neq W[1]$ for parameterized problems There is not fully PTAS for any W[1] problems.

A known problem that is W[1] hard but not considered yet NP hard is the Matrix VC dimension. This means that although this problem is not proven to be NP-complete doesn't have a FPTAS

Questions?

References & cool links

- Fundamentals of Parameterized Complexity
  `Rodney G. Downey , Michael R. Fellows`
- Parameterized Complexity Wiki
- Frontiers of Parameterized Complexity on Youtube
  `Seminar on Thursdays, at 17.00 Bergen time (GMT+2)`

Elli Anastasiadi

elli19@ru.is