Introduction to parameterized complexity

Marco Gario

EMCL / TUD

July 11, 2011

Why?

We are interested in parameterized complexity because:

- Better insight into problems complexity;
- Tools for handling intractability (NP, PSPACE or undecidability) under "certain conditions";
- By-products that are useful in other areas of AI (eg. preprocessing)
- Many low-hanging fruits, since most of the work has been done/applied only on graphs! Implementations and experimental results are rare!

Why?

We are interested in parameterized complexity because:

- Better insight into problems complexity;
- Tools for handling intractability (NP, PSPACE or undecidability) under "certain conditions";
- By-products that are useful in other areas of AI (eg. preprocessing)
- Many low-hanging fruits, since most of the work has been done/applied only on graphs! Implementations and experimental results are rare!

Content

Outline

Basic definitions

Definition: Parameterized problem

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet and the second component is called parameter.

- We represent a parameterized problem instance as (x, k) and call $n = |x|$ the *size* of the instance x.
- This definition allows us to characterize the running time of our problems and group them into classes depending both on n and k.

Example:

Definition: p-SAT

Instance:	A propositional formula F
Parameter:	Total number p of variables in F (ie. $\|var(F)\|$)
Question:	Decide whether F is satisfiable.

Fixed-parameter tractable (1/3)

Definition: Fpt-algorithm

An algorithm A is an *fpt-algorithm* if given an input x and a parameter k, the running time of A is at most $f(k) * n^c$, for some constant c and a computable function f.

Definition: Fixed-parameter tractable (FPT)

A parameterized problem L is *fixed-parameter tractable* if $(x, k) \in L$ can be decided by an fpt-algorithm (ie. in time $O(f(k) * n^c)$).

The name comes from the idea that, once we fix k, the problem becomes tractable (ie. polynomial-time).

Fixed-parameter tractable (2/3)

Example

p-SAT is FPT, since it can be solved in $O(2^p * n^c)$: we try all possible assignments for the variables and test in polynomial time whether it is a satisfying assignment.

Compared to classical complexity analysis, the size of the instance has only polynomial influence:

Example

Consider SAT: a problem size is given by the sum of the sizes of all clauses. Given two CNF formulas F, F' of size 10 and 11 with 4 variables each, we have the following runtime T:

- SAT: $O(2^n)$, $T(F') \approx 2T(F)$
- p-SAT: $O(2^p * n^c)$, $T(F') \approx T(F)$

Fixed-parameter tractable (3/3)

- The function $f(k)$ can be any computable function (depending only on k): we could run into incredibly steep functions like $2^{2^{2^k}}$.
  - technically: the problem is still polynomial time;
  - practically: we expect "real" problems to have "nice" parameterizations.
- A possible parameterization is usually "obvious", but there might be a different (and harder to find) one that behaves better.
- Obviously, not all parameterizations give us problems in FPT!

Fixed-parameter tractable (3/3)

- The function $f(k)$ can be any computable function (depending only on k): we could run into incredibly steep functions like $2^{2^{2^k}}$.
  - technically: the problem is still polynomial time;
  - practically: we expect "real" problems to have "nice" parameterizations.
- A possible parameterization is usually "obvious", but there might be a different (and harder to find) one that behaves better.
- Obviously, not all parameterizations give us problems in FPT!

Defining classes

In classical complexity, we use polynomial time *reductions* to build classes and order them into a hierarchy.

Definition: fpt-reduction ([4])

Let L, L' be two parameterized languages. We say that $L \leq_{fpt} L'$ iff there is an fpt-algorithm A such that

$$A : (G, k) \mapsto (G', k'),$$

so that

- $k' \leq f(k)$, for some computable function f
- $(G, k) \in L$ iff $(G', k') \in L'$

If $L \leq_{fpt} L'$ and $L' \leq_{fpt} L$ we say that $L \equiv_{fpt} L'$.

The Hierarchy

We can create fpt-equivalence classes and compose them into a hierarchy ([4]):

$$FPT \subseteq W[1] \subseteq W[2] \subseteq ...W[i]... \subseteq W[SAT] \subseteq W[P] \subseteq XP$$

- XP is the only class for which we know that $FPT \subset XP$: problems in this class have worst-case runtime $O(n^{f(k)})$
- Showing equivalence among any of the other classes, would make the hierarchy collapse
- FPT is "believed" to be different from $W[1]$: we consider only FPT as a tractable class

W-Hierarchy (1/2)

Given an interpretation I, we call *weight* of I the number of variables mapped to true.
For example, $I_1 = \{x, \bar{y}, \bar{z}\}$ has weight 1 while $I_2 = \{x, y, z\}$ has weight 3.

- W[SAT] is Weighted SAT for formulas that are not restricted syntactically: ie. generic propositional formulas.
- For the other $W[i]$ levels ($i \in \mathbb{N}$) the canonical problems are quite artificial: we will not cover them (see [3], [4])
  W[P] is "related" to weighted satisfiability of circuits of unbounded depth
- W[1] and W[2] are well studied classes, for which many canonical examples are available: eg. ([3])
  - Independent Set and Short Non-deterministic Turing Machine acceptance are W[1]-complete
  - Dominating Set and Weighted CNF Satisfiability are W[2]-complete

The field of parameterized complexity has an historical focus on graphs. Nevertheless, we will present Short NTM acceptance and Weighted CNF SAT.

W-Hierarchy (1/2)

Given an interpretation I, we call *weight* of I the number of variables mapped to true.
For example, $I_1 = \{x, \bar{y}, \bar{z}\}$ has weight 1 while $I_2 = \{x, y, z\}$ has weight 3.

- W[SAT] is Weighted SAT for formulas that are not restricted syntactically: ie. generic propositional formulas.
- For the other $W[i]$ levels ($i \in \mathbb{N}$) the canonical problems are quite artificial: we will not cover them (see [3], [4])
  W[P] is "related" to weighted satisfiability of circuits of unbounded depth
- W[1] and W[2] are well studied classes, for which many canonical examples are available: eg. ([3])
  - Independent Set and Short Non-deterministic Turing Machine acceptance are W[1]-complete
  - Dominating Set and Weighted CNF Satisfiability are W[2]-complete

The field of parameterized complexity has an historical focus on graphs. Nevertheless, we will present Short NTM acceptance and Weighted CNF SAT.

W-Hierarchy (1/2)

Given an interpretation I, we call *weight* of I the number of variables mapped to true.
For example, $I_1 = \{x, \bar{y}, \bar{z}\}$ has weight 1 while $I_2 = \{x, y, z\}$ has weight 3.

- W[SAT] is Weighted SAT for formulas that are not restricted syntactically: ie. generic propositional formulas.
- For the other $W[i]$ levels ($i \in \mathbb{N}$) the canonical problems are quite artificial: we will not cover them (see [3], [4])
  W[P] is "related" to weighted satisfiability of circuits of unbounded depth
- W[1] and W[2] are well studied classes, for which many canonical examples are available: eg. ([3])
  - Independent Set and Short Non-deterministic Turing Machine acceptance are W[1]-complete
  - Dominating Set and Weighted CNF Satisfiability are W[2]-complete

The field of parameterized complexity has an historical focus on graphs. Nevertheless, we will present Short NTM acceptance and Weighted CNF SAT.

W-Hierarchy (1/2)

Given an interpretation I, we call *weight* of I the number of variables mapped to true.
For example, $I_1 = \{x, \bar{y}, \bar{z}\}$ has weight 1 while $I_2 = \{x, y, z\}$ has weight 3.

- W[SAT] is Weighted SAT for formulas that are not restricted syntactically: ie. generic propositional formulas.
- For the other $W[i]$ levels ($i \in \mathbb{N}$) the canonical problems are quite artificial: we will not cover them (see [3], [4])
  W[P] is "related" to weighted satisfiability of circuits of unbounded depth
- W[1] and W[2] are well studied classes, for which many canonical examples are available: eg. ([3])
  - Independent Set and Short Non-deterministic Turing Machine acceptance are W[1]-complete
  - Dominating Set and Weighted CNF Satisfiability are W[2]-complete

The field of parameterized complexity has an historical focus on graphs. Nevertheless, we will present Short NTM acceptance and Weighted CNF SAT.

W-Hierarchy (2/2)

Definition: Short NTM Acceptence

Instance: A (single-tape) Non-deterministic Turing Machine M

Parameter: Maximum number of steps k

Question: Does M have an accepting computation in $\leq k$ steps?

Definition: Weighted CNF SAT

Instance: A CNF formula F

Parameter: Weight k

Question: Does F have a satisfying assignment of weight exactly k?

W-Hierarchy (2/2)

Definition: Short NTM Acceptence

Instance: A (single-tape) Non-deterministic Turing Machine M

Parameter: Maximum number of steps k

Question: Does M have an accepting computation in $\leq k$ steps?

Definition: Weighted CNF SAT

Instance: A CNF formula F

Parameter: Weight k

Question: Does F have a satisfying assignment of weight exactly k?

Outline

Vertex cover

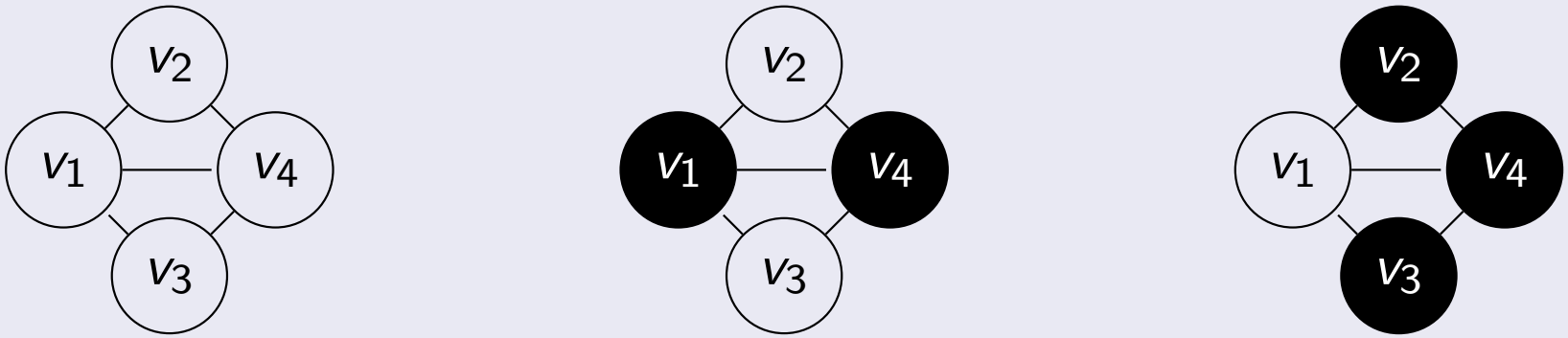
k-Vertex cover is one of the best known FPT problems.

Definition: Vertex Cover

Given a graph $G = (V, E)$, we call $C = \{v_1, .., v_n\} \subset V$ a *vertex cover* of G iff for all $e \in E$ there exists a $v_i \in C$ s.t. $v_i \in e$. In other words, we have a vertex $v_i \in C$ as "representative" of each edge in E. We call $|C|$ the size of the vertex cover.

Example

Consider $G = (V, E)$:



Then $C_1 = \{v_1, v_4\}$ and $C_2 = \{v_2, v_3, v_4\}$ are two different vertex covering for G of size, respectively, 2 and 3.

Vertex cover

k-Vertex cover is one of the best known FPT problems.

Definition: Vertex Cover

Given a graph $G = (V, E)$, we call $C = \{v_1, .., v_n\} \subset V$ a *vertex cover* of G iff for all $e \in E$ there exists a $v_i \in C$ s.t. $v_i \in e$. In other words, we have a vertex $v_i \in C$ as "representative" of each edge in E. We call $|C|$ the size of the vertex cover.

Definition: k-Vertex Cover

Instance: A graph $G = (V, E)$

Parameter: Upper-bound on the size of the cover k

Question: Does G have a vertex cover of size $\leq k$?

Bounded search (1/2)

Vertex cover can be trivially solved in $O(2^k)$ (eg. [5]):

Algorithm: vc(G,k)

Data: A graph $G = (V, E)$ an integer k
Result: A vertex cover C of size k
begin
    if $|E|==0$ **then return** true
    if $k==0$ **then return** false
    Pick an edge $e = (v1, v2)$
    if *vc(G[V\{v1}],k-1)* **then**
        $C = C \cup \{v1\}$ **return** *true*
    if *vc(G[V\{v2}],k-1)* **then**
        $C = C \cup \{v2\}$ **return** *true*
    return *false*
end

Bounded search (1/2)

Vertex cover can be trivially solved in $O(2^k)$ (eg. [5]):

Algorithm: vc(G,k)

Data: A graph $G = (V, E)$ an integer k
Result: A vertex cover C of size k
begin
    if $|E|$==*0* **then return** true
    if k==*0* **then return** false
    Pick an edge $e = (v1, v2)$
    if *vc(*$G[V \setminus \{v1\}]$*,k-1)* **then**
        $C = C \cup \{v1\}$ **return** *true*
    if *vc(*$G[V \setminus \{v2\}]$*,k-1)* **then**
        $C = C \cup \{v2\}$ **return** *true*
    return *false*
end

Bounded search (2/2)

- Bounding the solution size is an easy way (but not the only) to define a bounded search

$\rightarrow$ find a good/small branching rule: eg. Vertex Cover can be solved in $O(1.2738^k)$ ([2])

- Bounded search allows a trivial parallelization of the subtrees ([1])
- Many FPT results have been proved through bounded search (eg. strong Horn-backdoor detection)

Kernelization (1/2)

Even if "polynomially", the size of the problem influences our runtime: Can we reduce our problem to a smaller one?

Definition: Kernelization

Let (x, k) be a parameterized problem, a reduction to a problem kernel (or *kernelization*) is obtained by replacing an instance (x, k) by a reduced instance (x', k') (called problem kernel) such that:

- $k' \leq k$
- $|x'| \leq g(k)$ (for some g depending only on k)
- $(x, k) \in L$ iff $(x', k') \in L$
- (x', k') must be computable in polynomial time

Theorem: ([5])

A problem is FPT iff there exists a kernelization for it

Kernelization (2/2)

Example Kernelization for Vertex Cover

A possible kernelization is based on the following rules:

- VC1 Remove all isolated vertices
- VC2 For degree-1 vertices, put the neighbor node in the cover
- VC3 If there is a vertex of degree at least $k+1$, put the vertex in the cover

Rules VC2-3 decrease the value of k by one! After reaching a fix point with rule VC1-3, if the remaining graph has more than k^2 vertices, the instance is unsatisfiable.

The previous example is a kernelization of size $O(k^2)$, but better and more sophisticated kernelizations are possible.

Many others

Other (newer) methods are:

- Iterative compression: An algorithm that, given a problem x and a solution of size k, either calculates a smaller solution or proves that the given solution is of minimum size
- Color coding: Randomly color the "vertices" of the graph with an amount of color that is related to k, test if the result matches some characteristic: the probability is related to k
- Memoization: Once the problem and parameter are "small" enough, use a precomputed solution
- Treewidth

Many others

Other (newer) methods are:

- Iterative compression: An algorithm that, given a problem x and a solution of size k, either calculates a smaller solution or proves that the given solution is of minimum size
- Color coding: Randomly color the "vertices" of the graph with an amount of color that is related to k, test if the result matches some characteristic: the probability is related to k
- Memoization: Once the problem and parameter are "small" enough, use a precomputed solution
- Treewidth

Many others

Other (newer) methods are:

- Iterative compression: An algorithm that, given a problem x and a solution of size k, either calculates a smaller solution or proves that the given solution is of minimum size
- Color coding: Randomly color the "vertices" of the graph with an amount of color that is related to k, test if the result matches some characteristic: the probability is related to k
- Memoization: Once the problem and parameter are "small" enough, use a precomputed solution
- Treewidth

Many others

Other (newer) methods are:

- Iterative compression: An algorithm that, given a problem x and a solution of size k, either calculates a smaller solution or proves that the given solution is of minimum size
- Color coding: Randomly color the "vertices" of the graph with an amount of color that is related to k, test if the result matches some characteristic: the probability is related to k
- Memoization: Once the problem and parameter are "small" enough, use a precomputed solution
- Treewidth

Outline

FPT of SAT

- Chapter 13 of the Handbook of SAT (Samer and Szeider) is dedicated to the problem
- Definition: $\pi(F) = k$, for a formula F is a parameterization
- Define classes as $\mathcal{C}_k^\pi$ for all problems such that $\pi(F) \leq k$

$\rightarrow$ Hierarchy: $\mathcal{C}_0^\pi \subseteq \mathcal{C}_1^\pi \subseteq \mathcal{C}_2^\pi ...$

- We want problems for which membership and satisfiability in $\mathcal{C}_k^\pi$ can be decided in polynomial time
- Definition: $VER(\pi)$ is the parameterized problem of checking whether $\pi(F) \leq k$
- Definition: $SAT(\pi)$ is the parameterized problem that asks whether F with $\pi(F) \leq k$ is satisfiable
- Usually $VER(\pi)$ returns a witness that we can use for $SAT(\pi)$

Backdoors (1/2)

We quickly refresh the definition of backdoor ([6]):

Definition: Subsolver

We call an algorithm C a subsolver if, given an input formula F:

- **Tricotomy:** C either rejects the input F, or "determines" F correctly (as unsatisfiable or satisfiable, returning a solution if satisfiable),
- **Efficiency:** C runs in polynomial time,
- **Trivial solvability:** C can determine if F is trivially true or trivially false,
- **Self-reducibility:** if C determines F, then for any assignment v of the variable x C determines $F|_{\{x \mapsto v\}}$

where $F|_J$ indicates the reduct of F w.r.t. the (partial) interpretation J, that is the formula obtained by replacing each variable v in F with $J(v)$.

Backdoors (2/2)

Definition: Strong C-Backdoor

A non-empty subset B of the variables of the formula F is a *strong backdoor* w.r.t. the subsolver C for F iff **for all** interpretations $J : B \to \{\top, \bot\}$, C returns a satisfying assignment or concludes unsatisfiability of $F|_J$.

We call $\mathbf{b}_C$ the parameter measuring the size of the smallest strong C-backdoor:

- $VER(\mathbf{b}_C)$ and $SAT(\mathbf{b}_C)$ for $C \in \{Horn, 2SAT\}$ are FPT, via reduction to Vertex Cover
- $VER(\mathbf{b}_{RHorn})$ is W[1]-hard

Recall: A C-backdoor set B is called *smallest* if it is the set of minimal cardinality among all other C-backdoor sets.

Backdoors (2/2)

Definition: Strong C-Backdoor

A non-empty subset B of the variables of the formula F is a *strong backdoor* w.r.t. the subsolver C for F iff **for all** interpretations $J : B \to \{\top, \bot\}$, C returns a satisfying assignment or concludes unsatisfiability of $F|_J$.

We call $\mathbf{b}_C$ the parameter measuring the size of the smallest strong C-backdoor:

- $VER(\mathbf{b}_C)$ and $SAT(\mathbf{b}_C)$ for $C \in \{Horn, 2SAT\}$ are FPT, via reduction to Vertex Cover
- $VER(\mathbf{b}_{RHorn})$ is W[1]-hard

Recall: A C-backdoor set B is called *smallest* if it is the set of minimal cardinality among all other C-backdoor sets.

Treewidth (1/3)

Definition: Tree decomposition

A *tree decomposition* of a graph $G = (V, E)$ is a tree $T = (V', E')$ and a labeling function $\chi : V' \to 2^V$ associating to every node $t \in V'$ a set of vertices $\chi(t)$ such that:

- Every vertex in V occurs in some $\chi(t)$
- For every edge $(x, y) \in E$ there is a $\chi(t)$ that contains both x and y
- If $\chi(t_1)$ and $\chi(t_2)$ both contain x, then each $\chi(t_3)$ contains x if t_3 lies on the unique path from t_1 to t_2

The *width* of a tree decomposition is the maximum size of $\chi(t) - 1$ $(\forall t \in V')$

Definition: Treewidth

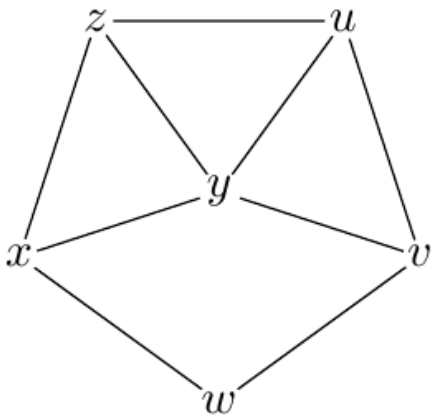
The *treewidth* of a graph is the minimum width among all its possible tree decompositions

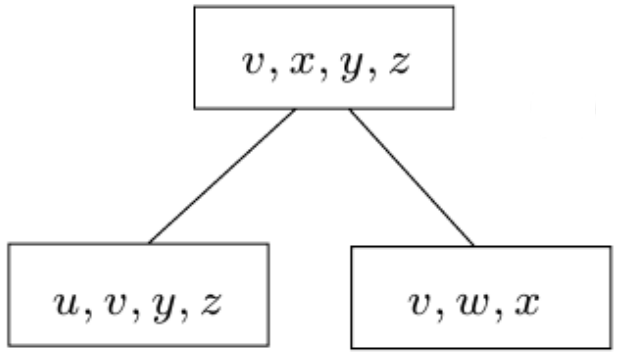
Treewidth (2/3)

Definition: Primal graph

The *primal graph* $G = (V, E)$ of a CNF formula F is obtained by associating each variable $i \in var(F)$ with a vertex $v_i \in V$ and adding an edge $e = (v_i, v_j) \in E$ iff $\exists C \in F . i, j \in var(C)$

Example for
$F = (\neg u \vee z) \wedge (u \vee \neg v \vee \neg y) \wedge (v \vee \neg w) \wedge (w \vee \neg x) \wedge (x \vee y \vee \neg z)$





Computing the treewidth is NP-hard, but the parameterized version is FPT (even linear).

Treewidth (3/3)

Definition: Dual graph

The *dual graph* $G = (V, E)$ of a CNF formula F is obtained by associating each clause $C_i \in F$ with a vertex $v_i \in V$ and adding an edge $e = (v_i, v_j) \in E$ iff $var(C_i) \cap var(C_j) \neq \emptyset$

Definition: Incidence graph

The *incidence graph* $G = (C \cup V, E)$ of a CNF formula F is a bipartite graph obtained by associating each clause $C_i \in F$ with a vertex $c_i \in C$ and each variable $j \in var(F)$ with a vertex $v_j \in V$. Edges are added iff a variable appears in a clause: $e = (c_i, v_j) \in E$ iff $j \in var(C_i)$.

- We call tw, tw^* and tw^d respectively the treewidth of the primal, incidence and dual graph.
- $VER(tw), VER(tw^*)$ and $VER(tw^d)$ are linear time
- $SAT(tw), SAT(tw^*)$ and $SAT(tw^d)$ are FPT

Treewidth (3/3)

Definition: Dual graph

The *dual graph* $G = (V, E)$ of a CNF formula F is obtained by associating each clause $C_i \in F$ with a vertex $v_i \in V$ and adding an edge $e = (v_i, v_j) \in E$ iff $var(C_i) \cap var(C_j) \neq \emptyset$

Definition: Incidence graph

The *incidence graph* $G = (C \cup V, E)$ of a CNF formula F is a bipartite graph obtained by associating each clause $C_i \in F$ with a vertex $c_i \in C$ and each variable $j \in var(F)$ with a vertex $v_j \in V$. Edges are added iff a variable appears in a clause: $e = (c_i, v_j) \in E$ iff $j \in var(C_i)$.

- We call tw, tw^* and tw^d respectively the treewidth of the primal, incidence and dual graph.
- $VER(tw), VER(tw^*)$ and $VER(tw^d)$ are linear time
- $SAT(tw), SAT(tw^*)$ and $SAT(tw^d)$ are FPT

SAT related problems in FPT

- 3SAT parameterized by the upperbound of the weight of the solution (weighted-SAT requires the *exact* weight of the solution)
- SAT parameterized by the maximum deficiency ($md = max_{F' \subset F} d(F')$ with $d(F') = |F| - |var(F)|$)
- Max-SAT parameterized by the number of satisfied clauses
- #SAT parameterized by the size of the smallest deletion Cluster-backdoor (A cluster formula is a disjoint variable union of hitting formulas, that is a formula in which each pair of clauses clash in at least one literal)
- #SAT parameterized by the treewidth of the incidence, primal and dual graph

Outline

Take home message

- There are tools that give us a more fine-grained analysis of the complexity of a problem: NP-complete $\neq$ "intractable"
- Parameterized complexity (and FPT in particular) are not just theoretical concepts, but they provide tools to better deal with complexity
- If we have many parameterizations of a problem, we can partition our problem space into areas that are "tractable" and use NP algorithms only when needed
- Much work has been done on graphs, but the theory can be applied to anything

Thank you

Questions?

References I

J Cheetham.
Solving large FPT problems on coarse-grained parallel machines.
Journal of Computer and System Sciences, 67(4):691–706, December 2003.

Jianer Chen, Iyad A Kanj, and Ge Xia.
LNCS 4162 - Improved Parameterized Upper Bounds for Vertex Cover.

Most, pages 238–249, 2006.

R. Downey.
Parameterized complexity for the skeptic.
18th IEEE Annual Conference on Computational Complexity, 2003. Proceedings., pages 147–168.

References II


J. Flum and Martin Grohe.
Parameterized complexity theory.
Springer-Verlag New York Inc, 2006.


F. Huffner, R. Niedermeier, and S. Wernicke.
Techniques for Practical Fixed-Parameter Algorithms.
The Computer Journal, 51(1):7–25, March 2007.


Ryan Williams, C.P. Gomes, and Bart Selman.
Backdoors to typical case complexity.
In *International joint conference on Artificial Intelligence*, volume 18, pages 1173–1178. Citeseer, 2003.

W[t]

We define the following classes of formulas, for $t \geq 0$ and $d \geq 1$:

- $\Gamma_{0,d}$ as a conjunction of d literals
- $\Delta_{0,d}$ as a disjunction of d literals
- $\Gamma_{t+1,d}$ (resp. $\Delta_{t+1,d}$) is defined as a conjunction (disjunction) of arbitrarily many $\Delta_{t,d}$ ($\Gamma_{t,d}$)

We alternate conjunction and disjunctions. CNF is defined as $\Gamma_{2,1}$: a conjunction of disjunction of "conjunctions of single literals".

Definition: p-WSAT(A)

Instance: A formula $F \in A$

Parameter: Weight k

Question: Does F have a satisfying assignment of weight exactly k?

Definition

W[t] p-WSAT($\Gamma_{t,d}$) is W[t]-complete for $d \geq 0$