Parameterized Algorithms and Complexity

Dániel Marx

Institute for Computer Science and Control, Hungarian Academy of Sciences (MTA SZTAKI)
Budapest, Hungary

PCSS 2017
Vienna, Austria
September 1, 2017
Outline

Goals of this talk:

1. A brief introduction to the world of parameterized algorithms.
   - Specific techniques (randomization, treewidth, kernelization, etc.) in later talks.

   - More complexity results based on ETH and SETH in later talks.
Parameterized problems

Main idea

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

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

Main idea

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

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

What can be the parameter $k$?

- The size $k$ of the solution we are looking for.
- The maximum degree of the input graph.
- The dimension of the point set in the input.
- The length of the strings in the input.
- The length of clauses in the input Boolean formula.
- ...
Parameterized complexity

**Problem:**

**Input:**
Graph $G$, integer $k$

**Question:**
Is it possible to cover the edges with $k$ vertices?

**Complexity:**
NP-complete

---

**Problem:**

**Input:**
Graph $G$, integer $k$

**Question:**
Is it possible to find $k$ independent vertices?

**Complexity:**
NP-complete
Parameterized complexity

**Problem:**

**Input:** Graph $G$, integer $k$

**Question:**

Is it possible to cover the edges with $k$ vertices?

Is it possible to find $k$ independent vertices?

**Complexity:**

**NP-complete**

**Brute force:**

$O(n^k)$ possibilities

---

**VERTEX COVER**

**INDEPENDENT SET**

Graph $G$, integer $k$

Graph $G$, integer $k$
Parameterized complexity

**Problem:**

**Input:** Graph $G$, integer $k$

**Question:** Is it possible to cover the edges with $k$ vertices? Is it possible to find $k$ independent vertices?

**Complexity:**

- **NP-complete**
- **Brute force:** $O(n^k)$ possibilities
  - $O(2^k n^2)$ algorithm exists 😊
  - No $n^{o(k)}$ algorithm known 😞

**Parameterized complexity**
Bounded search tree method

Algorithm for **Vertex Cover**:

\[ e_1 = u_1 v_1 \]
Bounded search tree method

Algorithm for **Vertex Cover**: 

\[ e_1 = u_1 v_1 \]
Bounded search tree method

Algorithm for **Vertex Cover**:

\[ e_1 = u_1 v_1 \]

\[ e_2 = u_2 v_2 \]
Bounded search tree method

Algorithm for **Vertex Cover:**

\[ e_1 = u_1 v_1 \]

\[ e_2 = u_2 v_2 \]
Bounded search tree method

Algorithm for Vertex Cover:

$e_1 = u_1v_1$

$e_2 = u_2v_2$

Height of the search tree $\leq k \Rightarrow$ at most $2^k$ leaves $\Rightarrow 2^k \cdot n^{O(1)}$ time algorithm.
Fixed-parameter tractability

Main definition

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

Examples of NP-hard problems that are FPT:

- Finding a vertex cover of size $k$.
- Finding a path of length $k$.
- Finding $k$ disjoint triangles.
- Drawing the graph in the plane with $k$ edge crossings.
- Finding disjoint paths that connect $k$ pairs of points.

...
Fixed-parameter tractability

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

Examples of NP-hard problems that are FPT:
- Finding a vertex cover of size $k$.
- Finding a path of length $k$.
- Finding $k$ disjoint triangles.
- Drawing the graph in the plane with $k$ edge crossings.
- Finding disjoint paths that connect $k$ pairs of points.
- ...
FPT techniques

Bounded-depth search trees

Kernelization

Color coding

Algebraic techniques

Iterative compression

Treewidth
W[1]-hardness

Negative evidence similar to NP-completeness. If a problem is **W[1]-hard**, then the problem is not FPT unless FPT=W[1].

Some W[1]-hard problems:
- Finding a clique/independent set of size $k$.
- Finding a dominating set of size $k$.
- Finding $k$ pairwise disjoint sets.
- ...
The study of parameterized complexity was initiated by Downey and Fellows in the early 90s.
First monograph in 1999.
By now, strong presence in most algorithmic conferences.
Shift of focus

qualitative question

FPT or W[1]-hard?
Shift of focus

**FPT or W[1]-hard?**

**qualitative question**
- What is the best possible multiplier $f(k)$ in the running time $f(k) \cdot n^{O(1)}$?
- What is the best possible exponent $g(k)$ in the running time $f(k) \cdot n^{g(k)}$?

**quantitative question**
- $2^k$?
- $1.0001^k$?
- $2^{\sqrt{k}}$?
- $n^{O(k)}$?
- $n^\log k$?
- $n^{\log \log k}$?

2
11
Single-exponential running time

The following problems can be solved in time $2^{O(k)} \cdot n^{O(1)}$, but (assuming ETH) cannot be solved in time $2^{o(k)} \cdot n^{O(1)}$:

- Vertex Cover
- Longest Cycle
- Feedback Vertex Set
- Multiway Cut
- Odd Cycle Transversal
- Steiner Tree
- ...

Seems to be the natural behavior of FPT problems?
The race for better FPT algorithms

Double exponential

Tower of exponentials

"Slightly super-exponential"

Single exponential

Subexponential
Graph Minors Theory

Theory of graph minors developed in the monumental series

Graph Minors I–XXIII.
*J. Combin. Theory, Ser. B*
1983–2012

Neil Robertson  Paul Seymour

- Structure theory of graphs excluding minors (and much more).
- Galactic combinatorial bounds and running times.
- Important early influence for parameterized algorithms.

[figure by Felix Reidl]
Disjoint paths

**\( k \)-Disjoint Paths**

Given a graph \( G \) and pairs of vertices \((s_1, t_1), \ldots, (s_k, t_k)\), find pairwise vertex-disjoint paths \( P_1, \ldots, P_k \) such that \( P_i \) connects \( s_i \) and \( t_i \).
Disjoint paths

**k-Disjoint Paths**
Given a graph $G$ and pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$, find pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that $P_i$ connects $s_i$ and $t_i$. 

![Diagram of disjoint paths](image-url)
Disjoint paths

**k-Disjoint Paths**
Given a graph $G$ and pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$, find pairwise vertex-disjoint paths $P_1, \ldots, P_k$ such that $P_i$ connects $s_i$ and $t_i$.

- NP-hard, but FPT parameterized by $k$: can be solved in time $f(k)n^3$ for some horrible function $f(k)$ [Robertson and Seymour].
- More “efficient” algorithm where $f(k)$ is only quadruple exponential [Kawarabayashi and Wollan 2010].
- The Polynomial Excluded Grid Theorem improves this to triple exponential [Chekuri and Chuzhoy 2014].
- Double-exponential is possible on planar graphs [Adler et al. 2011].

Open: can we have a $2^{kO(1)} \cdot n^{O(1)}$ time algorithm?
**Edge Clique Cover**

**Edge Clique Cover:** Given a graph $G$ and an integer $k$, cover the edges of $G$ with at most $k$ cliques.

(the cliques need not be edge disjoint)

**Equivalently:** can $G$ be represented as an intersection graph over a $k$ element universe?
**Edge Clique Cover**

**Edge Clique Cover:** Given a graph $G$ and an integer $k$, cover the edges of $G$ with at most $k$ cliques. (the cliques need not be edge disjoint)

**Equivalently:** can $G$ be represented as an intersection graph over a $k$ element universe?

![Diagram with 6 cliques]
**Edge Clique Cover**

**Edge Clique Cover:** Given a graph $G$ and an integer $k$, cover the edges of $G$ with at most $k$ cliques. (the cliques need not be edge disjoint)

**Equivalently:** can $G$ be represented as an intersection graph over a $k$ element universe?

![Diagram of 5 cliques]
**Edge Clique Cover**

**Edge Clique Cover**: Given a graph $G$ and an integer $k$, cover the edges of $G$ with at most $k$ cliques. (the cliques need not be edge disjoint)

- Can be solved in time $2^{2^{O(k)}} \cdot n^{O(1)}$ — double exponential dependence on $k$.
- Assuming ETH, double-exponential dependence on $k$ cannot be avoided [Cygan, Pilipczuk, Pilipczuk 2013].
Slightly superexponential algorithms

Running time of the form $2^{O(k \log k)} \cdot n^{O(1)}$ appear naturally in parameterized algorithms usually because of one of two reasons:

1. Branching into $k$ directions at most $k$ times explores a search tree of size $k^k = 2^{O(k \log k)}$.

2. Trying $k! = 2^{O(k \log k)}$ permutations of $k$ elements (or partitions, matchings, ...)

Can we avoid these steps and obtain $2^{O(k)} \cdot n^{O(1)}$ time algorithms?
Slightly superexponential algorithms

The improvement to $2^{O(k)}$ often required significant new ideas:

**k-Path:**

1. $2^{O(k \log k)} \cdot n^{O(1)}$ using **representative sets** [Monien 1985]
2. $2^{O(k)} \cdot n^{O(1)}$ using **color coding** [Alon, Yuster, Zwick 1995]

**Feedback Vertex Set:**

1. $2^{O(k \log k)} \cdot n^{O(1)}$ using **k-way branching** [Downey and Fellows 1995]
2. $2^{O(k)} \cdot n^{O(1)}$ using **iterative compression** [Guo et al. 2005]

**Planar Subgraph Isomorphism:**

1. $2^{O(k \log k)} \cdot n^{O(1)}$ using **tree decompositions** [Eppstein et al. 1995]
2. $2^{O(k)} \cdot n^{O(1)}$ using **sphere cut decompositions** [Dorn 2010]
The race for better FPT algorithms

- Double exponential: $2^{2^O(k)}$
- "Slightly super-exponential": $2^{O(k \log k)}$
- Single exponential: $2^{O(k)}$
- Subexponential: $2^{O(\sqrt{k \log k})}$
- Tower of exponentials: $2^{2^\ldots^{2^k}}$

- $f(k)$
Subexponential parameterized algorithms

There are two main domains where subexponential parameterized algorithms appear:

1. Some graph modification problems:
   - **Chordal Completion** [Fomin and Villanger 2013]
   - **Interval Completion** [Bliznets et al. 2016]
   - **Unit Interval Completion** [Bliznets et al. 2015]
   - **Feedback Arc Set in Tournaments** [Alon et al. 2009]
Subexponential parameterized algorithms

There are two main domains where subexponential parameterized algorithms appear:

1. Some graph modification problems:
   - **Chordal Completion** [Fomin and Villanger 2013]
   - **Interval Completion** [Bliznets et al. 2016]
   - **Unit Interval Completion** [Bliznets et al. 2015]
   - **Feedback Arc Set in Tournaments** [Alon et al. 2009]

2. “Square root phenomenon” for planar graphs and geometric objects: most NP-hard problems are easier and usually exactly by a square root factor.

Planar graphs

Geometric objects
**Chordal Completion**

**Definition:** A graph is **chordal** if it does not contain an induced cycle of length greater than 3.

**Chordal Completion:** Given a graph $G$ and an integer $k$, add at most $k$ edges to $G$ to make it a chordal graph.
**Definition:** A graph is **chordal** if it does not contain an induced cycle of length greater than 3.

**Chordal Completion:** Given a graph $G$ and an integer $k$, add at most $k$ edges to $G$ to make it a chordal graph.

**Lemma:** At least $k - 3$ edges are needed to make a $k$-cycle chordal.

**Proof:** By induction. $k = 3$ is trivial.
**Chordal Completion**

**Definition:** A graph is **chordal** if it does not contain an induced cycle of length greater than 3.

**Chordal Completion:** Given a graph $G$ and an integer $k$, add at most $k$ edges to $G$ to make it a chordal graph.

**Lemma:** At least $k - 3$ edges are needed to make a $k$-cycle chordal.

**Proof:** By induction. $k = 3$ is trivial.

\[
\begin{align*}
C_x & : x - 3 \text{ edges} \\
C_{k-x+2} & : k - x - 1 \text{ edges} \\
C_k & : (x-3) + (k-x-1) + 1 = k - 3 \text{ edges}
\end{align*}
\]
Chordal Completion

Algorithm:
- Find an induced cycle $C$ of length $\geq 4$ (can be done in polynomial time).
- If no such cycle exists $\Rightarrow$ Done!
- If $C$ has more than $k + 3$ vertices $\Rightarrow$ No solution!
- Otherwise, one of the

$$\binom{|C|}{2} - |C| \leq (k + 3)(k + 2)/2 - k = O(k^2)$$

missing edges has to be added $\Rightarrow$ Branch!

Size of the search tree is $k^{O(k)}$. 
Chordal Completion – more efficiently

**Definition:** Triangulation of a cycle.

**Lemma:** Every chordal supergraph of a cycle $C$ contains a triangulation of the cycle $C$.

**Lemma:** The number of ways a cycle of length $k$ can be triangulated is exactly the $(k - 2)$-nd Catalan number

$$C_{k-2} = \frac{1}{k-1} \binom{2(k-2)}{k-2} \leq 4^{k-3}.$$
Algorithm:

- Find an induced cycle $C$ of length at least 4 (can be done in polynomial time).
- If no such cycle exists $\Rightarrow$ Done!
- If $C$ has more than $k + 3$ vertices $\Rightarrow$ No solution!
- Otherwise, one of the $\leq 4^{|C| - 3}$ triangulations has to be in the solution $\Rightarrow$ Branch!

Claim: Search tree has at most $T_k = 4^k$ leaves.

Proof: By induction. Number of leaves is at most

$$T_k \leq 4^{|C| - 3} \cdot T_{k-(|C|-3)} \leq 4^{|C| - 3} \cdot 4^{k-(|C|-3)} = 4^k.$$
Subexpontential algorithms on planar graphs

Most NP-hard problems (e.g., 3-Coloring, Independent Set, Hamiltonian Cycle, Steiner Tree, etc.) remain NP-hard on planar graphs,\(^1\) so what do we mean by “easier”?\(^2\)

---

\(^1\)Notable exception: Max Cut is in P for planar graphs.
Subexponential algorithms on planar graphs

Most NP-hard problems (e.g., \textbf{3-Coloring}, \textbf{Independent Set}, \textbf{Hamiltonian Cycle}, \textbf{Steiner Tree}, etc.) remain NP-hard on planar graphs,\(^1\) so what do we mean by “easier”?

The running time is still exponential, but significantly smaller:

\[
\begin{align*}
2^{O(n)} & \Rightarrow 2^{O(\sqrt{n})} \\
n^{O(k)} & \Rightarrow n^{O(\sqrt{k})} \\
2^{O(k)} \cdot n^{O(1)} & \Rightarrow 2^{O(\sqrt{k})} \cdot n^{O(1)}
\end{align*}
\]

\(^1\)Notable exception: \textbf{Max Cut} is in P for planar graphs.
Subexponential algorithms on planar graphs

The following problems can be solved in time $2^{O(\sqrt{k}\cdot\text{polylog}k)} \cdot n^{O(1)}$ on planar graphs:

- **Vertex Cover**
- **$k$-Path**
- **Independent Set**
- **Dominating Set**
- **Feedback Vertex Set**
- **Subset TSP**
- **Subgraph Isomorphism** for bounded degree connected patterns.
Subexpontential algorithms on planar graphs

The following problems can be solved in time $n^{O(k)}$ on general graphs, which can be improved to $f(k)n^{O(\sqrt{k})}$ on planar graphs:

- **Distance-$d$ Independent Set** on planar graphs
- **Distance-$d$ Dominating Set** on planar graphs
- **Strongly Connected Steiner Subgraph** on directed planar graphs
- **Independent Set** for unit disks in the plane
Multiway Cut

**k-Terminal Cut (aka Multiway Cut)**

- **Input:** A graph $G$, an integer $p$, and a set $T$ of $k$ terminals
- **Output:** A set $S$ of at most $p$ edges such that removing $S$ separates any two vertices of $T$

**Theorem**

NP-hard already for $k = 3$. 

---

**Orange diagram with red edges showing a cut**
Multiway Cut

**k-Terminal Cut (aka Multiway Cut)**

**Input:** A graph $G$, an integer $p$, and a set $T$ of $k$ terminals

**Output:** A set $S$ of at most $p$ edges such that removing $S$ separates any two vertices of $T$

**Theorem**

**Planar $k$-Terminal Cut** can be solved in time $2^{O(k)} \cdot n^{O(\sqrt{k})}$. 
Lower bounds

So far we have seen positive results: basic algorithmic techniques for fixed-parameter tractability.

What kind of negative results we have?

- Can we show that a problem (e.g., *Clique*) is not FPT?
- Can we show that a problem (e.g., *Vertex Cover*) has no algorithm with running time, say, $2^{o(k)} \cdot n^{O(1)}$?

- This would require showing that $P \neq NP$: if $P = NP$, then, e.g., $k$-Clique is polynomial-time solvable, hence FPT.
Lower bounds

So far we have seen positive results: basic algorithmic techniques for fixed-parameter tractability.

What kind of negative results we have?

- Can we show that a problem (e.g., \textit{Clique}) is not FPT?
- Can we show that a problem (e.g., \textit{Vertex Cover}) has no algorithm with running time, say, $2^{o(k)} \cdot n^{O(1)}$?

This would require showing that $P \neq NP$: if $P = NP$, then, e.g., \textit{k-Clique} is polynomial-time solvable, hence FPT.

Can we give some evidence for negative results?
Classical complexity

Nondeterministic Turing Machine (NTM): single tape, finite alphabet, finite state, head can move left/right only one cell. In each step, the machine can branch into an arbitrary number of directions. Run is successful if at least one branch is successful.

NP: The class of all languages that can be recognized by a polynomial-time NTM.

Polynomial-time reduction from problem \( P \) to problem \( Q \): a function \( \phi \) with the following properties:

- \( \phi(x) \) can be computed in time \( |x|^{O(1)} \),
- \( \phi(x) \) is a yes-instance of \( Q \) if and only if \( x \) is a yes-instance of \( P \).

Definition: Problem \( Q \) is NP-hard if any problem in NP can be reduced to \( Q \).

If an NP-hard problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time (i.e., \( P = NP \)).
Parameterized complexity

To build a complexity theory for parameterized problems, we need two concepts:

- An appropriate notion of reduction.
- An appropriate hypothesis.

Polynomial-time reductions are not good for our purposes.
Parameterized complexity

To build a complexity theory for parameterized problems, we need two concepts:

- An appropriate notion of reduction.
- An appropriate hypothesis.

Polynomial-time reductions are not good for our purposes.

**Example:** Graph $G$ has an independent set $k$ if and only if it has a vertex cover of size $n - k$.

$\Rightarrow$ Transforming an **Independent Set** instance $(G, k)$ into a **Vertex Cover** instance $(G, n - k)$ is a correct polynomial-time reduction.

However, **Vertex Cover** is FPT, but **Independent Set** is not known to be FPT.
Parameterized reduction

**Definition**

Parameterized reduction from problem $P$ to problem $Q$: a function $\phi$ with the following properties:

- $\phi(x)$ can be computed in time $f(k) \cdot |x|^{O(1)}$, where $k$ is the parameter of $x$,
- $\phi(x)$ is a yes-instance of $Q$ $\iff$ $x$ is a yes-instance of $P$.
- If $k$ is the parameter of $x$ and $k'$ is the parameter of $\phi(x)$, then $k' \leq g(k)$ for some function $g$.

**Fact:** If there is a parameterized reduction from problem $P$ to problem $Q$ and $Q$ is FPT, then $P$ is also FPT.
Parameterized reduction

**Definition**

Parameterized reduction from problem $P$ to problem $Q$: a function $\phi$ with the following properties:

- $\phi(x)$ can be computed in time $f(k) \cdot |x|^{O(1)}$, where $k$ is the parameter of $x$,
- $\phi(x)$ is a yes-instance of $Q \iff x$ is a yes-instance of $P$.
- If $k$ is the parameter of $x$ and $k'$ is the parameter of $\phi(x)$, then $k' \leq g(k)$ for some function $g$.

**Fact:** If there is a parameterized reduction from problem $P$ to problem $Q$ and $Q$ is FPT, then $P$ is also FPT.

**Non-example:** Transforming an Independent Set instance $(G, k)$ into a Vertex Cover instance $(G, n - k)$ is not a parameterized reduction.

**Example:** Transforming an Independent Set instance $(G, k)$ into a Clique instance $(\overline{G}, k)$ is a parameterized reduction.
**Multicolored Clique**

A useful variant of **Clique**:

**Multicolored Clique**: The vertices of the input graph $G$ are colored with $k$ colors and we have to find a clique containing one vertex from each color.

(or **Partitioned Clique**)

---

**Theorem**

There is a parameterized reduction from **Clique** to **Multicolored Clique**.
**Multicolored Clique**

**Theorem**

There is a parameterized reduction from **Clique** to **Multicolored Clique**.

Create $G'$ by replacing each vertex $v$ with $k$ vertices, one in each color class. If $u$ and $v$ are adjacent in the original graph, connect all copies of $u$ with all copies of $v$.

$k$-clique in $G \iff$ multicolored $k$-clique in $G'$. 
**Multicolored Clique**

**Theorem**

There is a parameterized reduction from **Clique** to **Multicolored Clique**.

Create $G'$ by replacing each vertex $v$ with $k$ vertices, one in each color class. If $u$ and $v$ are adjacent in the original graph, connect all copies of $u$ with all copies of $v$.

$k$-clique in $G$ $\iff$ multicolored $k$-clique in $G'$.

**Similarly:** reduction to **Multicolored Independent Set**.
**Theorem**

There is a parameterized reduction from **Multicolored Independent Set** to **Dominating Set**.

**Proof:** Let $G$ be a graph with color classes $V_1, \ldots, V_k$. We construct a graph $H$ such that $G$ has a multicolored $k$-clique iff $H$ has a dominating set of size $k$.

```
\begin{align*}
V_1 &= \begin{bmatrix}
    x_1 & y_1 \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet
\end{bmatrix} \\
V_2 &= \begin{bmatrix}
    x_2 & y_2 \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet
\end{bmatrix} \\
V_k &= \begin{bmatrix}
    x_k & y_k \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet \\
    \bullet & \bullet
\end{bmatrix}
\end{align*}
```

- The dominating set has to contain one vertex from each of the $k$ cliques $V_1, \ldots, V_k$ to dominate every $x_i$ and $y_i$. 

```
Dominating Set

Theorem

There is a parameterized reduction from Multicolored Independent Set to Dominating Set.

Proof: Let $G$ be a graph with color classes $V_1, \ldots, V_k$. We construct a graph $H$ such that $G$ has a multicolored $k$-clique iff $H$ has a dominating set of size $k$.

- The dominating set has to contain one vertex from each of the $k$ cliques $V_1, \ldots, V_k$ to dominate every $x_i$ and $y_i$.
- For every edge $e = uv$, an additional vertex $w_e$ ensures that these selections describe an independent set.
Variants of Dominating Set

- **Dominating Set**: Given a graph, find $k$ vertices that dominate every vertex.
- **Red-Blue Dominating Set**: Given a bipartite graph, find $k$ vertices on the red side that dominate the blue side.
- **Set Cover**: Given a set system, find $k$ sets whose union covers the universe.
- **Hitting Set**: Given a set system, find $k$ elements that intersect every set in the system.

All of these problems are equivalent under parameterized reductions, hence at least as hard as Clique.
Hard problems

Hundreds of parameterized problems are known to be at least as hard as \textbf{Clique}:

- \textbf{Independent Set}
- \textbf{Set Cover}
- \textbf{Hitting Set}
- \textbf{Connected Dominating Set}
- \textbf{Independent Dominating Set}
- \textbf{Partial Vertex Cover} parameterized by $k$
- \textbf{Dominating Set} in bipartite graphs
- \ldots

We believe that none of these problems are FPT.
Basic hypotheses

It seems that parameterized complexity theory cannot be built on assuming $P \neq \text{NP}$ – we have to assume something stronger.

Let us choose a basic hypothesis:

<table>
<thead>
<tr>
<th>Engineers’ Hypothesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$-CLIQUE cannot be solved in time $f(k) \cdot n^{O(1)}$.</td>
</tr>
</tbody>
</table>
Basic hypotheses

It seems that parameterized complexity theory cannot be built on assuming $P \neq NP$ – we have to assume something stronger.

Let us choose a basic hypothesis:

**Engineers’ Hypothesis**

$k$-CLIQUE cannot be solved in time $f(k) \cdot n^{O(1)}$.

**Theorists’ Hypothesis**

$k$-STEP HALTING PROBLEM (is there a path of the given NTM that stops in $k$ steps?) cannot be solved in time $f(k) \cdot n^{O(1)}$. 

Exponential Time Hypothesis (ETH) $n$-variable 3SAT cannot be solved in time $2^{o(n)}$. Which hypothesis is the most plausible?
Basic hypotheses

It seems that parameterized complexity theory cannot be built on assuming $P \neq NP$ – we have to assume something stronger.

Let us choose a basic hypothesis:

**Engineers’ Hypothesis**

$k$-CLIQUE cannot be solved in time $f(k) \cdot n^{O(1)}$.

**Theorists’ Hypothesis**

$k$-STEP HALTING PROBLEM (is there a path of the given NTM that stops in $k$ steps?) cannot be solved in time $f(k) \cdot n^{O(1)}$.

**Exponential Time Hypothesis (ETH)**

$n$-variable 3SAT cannot be solved in time $2^{o(n)}$.

Which hypothesis is the most plausible?
Basic hypotheses

It seems that parameterized complexity theory cannot be built on assuming $P \neq NP$ – we have to assume something stronger.

Let us choose a basic hypothesis:

**Engineers’ Hypothesis**

\[ k\text{-Clique} \text{ cannot be solved in time } f(k) \cdot n^{O(1)} \text{.} \]

**Theorists’ Hypothesis**

\[ k\text{-Step Halting Problem} \text{ (is there a path of the given NTM that stops in } k \text{ steps?) cannot be solved in time } f(k) \cdot n^{O(1)} \text{.} \]

**Exponential Time Hypothesis (ETH)**

\[ n\text{-variable 3SAT cannot be solved in time } 2^{o(n)} \text{.} \]

Which hypothesis is the most plausible?
Summary of complexity

- **Independent Set** and **k-Step Halting Problem** can be reduced to each other $\Rightarrow$ Engineers’ Hypothesis and Theorists’ Hypothesis are equivalent!

- **Independent Set** and **k-Step Halting Problem** can be reduced to **Dominating Set**.
Summary of complexity

- **Independent Set** and **k-Step Halting Problem** can be reduced to each other $\Rightarrow$ Engineers’ Hypothesis and Theorists’ Hypothesis are equivalent!

- **Independent Set** and **k-Step Halting Problem** can be reduced to **Dominating Set**.

- Is there a parameterized reduction from **Dominating Set** to **Independent Set**?
  - Probably not. Unlike in NP-completeness, where most problems are equivalent, here we have a hierarchy of hard problems.
    - **Independent Set** is W[1]-complete.
    - **Dominating Set** is W[2]-complete.

- Does not matter if we only care about whether a problem is FPT or not!
A **Boolean circuit** consists of input gates, negation gates, AND gates, OR gates, and a single output gate.

**Circuit Satisfiability**: Given a Boolean circuit $C$, decide if there is an assignment on the inputs of $C$ making the output true.
Boolean circuit

A **Boolean circuit** consists of input gates, negation gates, AND gates, OR gates, and a single output gate.

![Boolean circuit diagram]

**Circuit Satisfiability**: Given a Boolean circuit $C$, decide if there is an assignment on the inputs of $C$ making the output true.

**Weight of an assignment**: number of true values.

**Weighted Circuit Satisfiability**: Given a Boolean circuit $C$ and an integer $k$, decide if there is an assignment of weight $k$ making the output true.
**Weighted Circuit Satisfiability**

**Independent Set** can be reduced to **Weighted Circuit Satisfiability**:

```
x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_6 \quad x_7
```

- **Dominating Set** can be reduced to **Weighted Circuit Satisfiability**:

```
x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_6 \quad x_7
```

To express **Dominating Set**, we need more complicated circuits.
**Weighted Circuit Satisfiability**

**Independent Set** can be reduced to **Weighted Circuit Satisfiability**:

![Diagram](image1)

**Dominating Set** can be reduced to **Weighted Circuit Satisfiability**:

![Diagram](image2)

To express **Dominating Set**, we need more complicated circuits.
Depth and weft

The **depth** of a circuit is the maximum length of a path from an input to the output.

A gate is **large** if it has more than 2 inputs. The **weft** of a circuit is the maximum number of large gates on a path from an input to the output.

**Independent Set:** weft 1, depth 3

**Dominating Set:** weft 2, depth 2
The W-hierarchy

Let $C[t, d]$ be the set of all circuits having weft at most $t$ and depth at most $d$.

**Definition**

A problem $P$ is in the class $W[t]$ if there is a constant $d$ and a parameterized reduction from $P$ to *Weighted Circuit Satisfiability* of $C[t, d]$.

We have seen that *Independent Set* is in $W[1]$ and *Dominating Set* is in $W[2]$.

**Fact:** *Independent Set* is $W[1]$-complete.

**Fact:** *Dominating Set* is $W[2]$-complete.
The W-hierarchy

Let \( C[t, d] \) be the set of all circuits having weft at most \( t \) and depth at most \( d \).

**Definition**

A problem \( P \) is in the class \( W[t] \) if there is a constant \( d \) and a parameterized reduction from \( P \) to **Weighted Circuit Satisfiability** of \( C[t, d] \).

We have seen that **Independent Set** is in \( W[1] \) and **Dominating Set** is in \( W[2] \).

**Fact:** **Independent Set** is \( W[1] \)-complete.

**Fact:** **Dominating Set** is \( W[2] \)-complete.

If any \( W[1] \)-complete problem is FPT, then \( \text{FPT} = W[1] \) and every problem in \( W[1] \) is FPT.


\( \Rightarrow \) If there is a parameterized reduction from **Dominating Set** to **Independent Set**, then \( W[1] = W[2] \).
Weft is a term related to weaving cloth: it is the thread that runs from side to side in the fabric.
What did we learn, Palmer?

- The initial question: FPT or W[1]-hard?
- More refined question: what is the exact best possible running time?
- Surprising running times appear naturally.
- Using W[1]-hardness and parameterized reductions to give evidence that a problem is not FPT.
Postdoc positions available in parameterized algorithms and complexity!

Institute for Computer Science and Control
Hungarian Academy of Sciences
Budapest, Hungary