8. Parameterized intractability: the W-hierarchy

COMP6741: Parameterized and Exact Computation

Serge Gaspers$^{1,2}$

$^1$School of Computer Science and Engineering, UNSW Australia
$^2$Optimisation Research Group, NICTA

Semester 2, 2015
1. Reminder: Polynomial Time Reductions and NP-completeness

2. Parameterized Complexity Theory
   - Parameterized reductions
   - Parameterized complexity classes

3. Case studies

4. Further Reading
Outline

1. Reminder: Polynomial Time Reductions and NP-completeness

2. Parameterized Complexity Theory
   - Parameterized reductions
   - Parameterized complexity classes

3. Case studies

4. Further Reading
Polynomial-time reduction

**Definition 1**

A polynomial-time reduction from a decision problem \( \Pi_1 \) to a decision problem \( \Pi_2 \) is a polynomial-time algorithm, which, for any instance of \( \Pi_1 \) produces an equivalent instance of \( \Pi_2 \).

If there exists a polynomial-time reduction from \( \Pi_1 \) to \( \Pi_2 \), we say that \( \Pi_1 \) is polynomial-time reducible to \( \Pi_2 \) and write \( \Pi_1 \leq_P \Pi_2 \).
New polynomial-time algorithms via reductions

**Lemma 2**

If $\Pi_1, \Pi_2$ are decision problems such that $\Pi_1 \leq_P \Pi_2$, then $\Pi_2 \in P$ implies $\Pi_1 \in P$. 
NP-completeness

**Definition 3 (NP-hard)**

A decision problem $\Pi$ is **NP-hard** if $\Pi' \leq_p \Pi$ for every $\Pi' \in \text{NP}$.

**Definition 4 (NP-complete)**

A decision problem $\Pi$ is **NP-complete** (in NPC) if

1. $\Pi \in \text{NP}$, and
2. $\Pi$ is NP-hard.
Lemma 5

If $\Pi$ is a decision problem such that $\Pi' \leq_p \Pi$ for some NP-hard decision problem $\Pi'$, then $\Pi$ is NP-hard.

If, in addition, $\Pi \in \text{NP}$, then $\Pi \in \text{NPC}$.
Method to prove that a decision problem \( \Pi \) is NP-complete:

1. Prove \( \Pi \in \text{NP} \)
2. Prove \( \Pi \) is NP-hard.
   - Select a known NP-hard decision problem \( \Pi' \).
   - Describe an algorithm that transforms every instance \( I \) of \( \Pi' \) to an instance \( r(I) \) of \( \Pi \).
   - Prove that for each instance \( I \) of \( \Pi' \), we have that \( I \) is a Yes-instance of \( \Pi' \) \( \iff \) \( r(I) \) is a Yes-instance of \( \Pi \).
   - Show that the algorithm runs in polynomial time.
Outline

1. Reminder: Polynomial Time Reductions and NP-completeness

2. Parameterized Complexity Theory
   - Parameterized reductions
   - Parameterized complexity classes

3. Case studies

4. Further Reading
Main Parameterized Complexity Classes

$n$: instance size  
$k$: parameter  

\(P\): class of problems that can be solved in \(n^{O(1)}\) time  
\(\text{FPT}\): class of parameterized problems that can be solved in \(f(k) \cdot n^{O(1)}\) time  
\(W[\cdot]\): parameterized intractability classes  
\(\text{XP}\): class of parameterized problems that can be solved in \(f(k) \cdot n^{g(k)}\) time  
\(\text{“polynomial when } k \text{ is a constant”}\)

\(P \subseteq \text{FPT} \subseteq W[1] \subseteq W[2] \cdots \subseteq W[P] \subseteq \text{XP}\)

\textbf{Note:} We assume that \(f\) is \textit{computable} and \textit{non-decreasing}. 
Polynomial-time reductions for parameterized problems?

A vertex cover in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every edge of $G$ has an endpoint in $S$.

**Vertex Cover**
- **Input:** Graph $G$, integer $k$
- **Parameter:** $k$
- **Question:** Does $G$ have a vertex cover of size $k$?

An independent set in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that there is no edge $uv \in E$ with $u, v \in S$.

**Independent Set**
- **Input:** Graph $G$, integer $k$
- **Parameter:** $k$
- **Question:** Does $G$ have an independent set of size $k$?
A vertex cover in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every edge of $G$ has an endpoint in $S$.

**Vertex Cover**
- **Input:** Graph $G$, integer $k$
- **Parameter:** $k$
- **Question:** Does $G$ have a vertex cover of size $k$?

An independent set in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that there is no edge $uv \in E$ with $u, v \in S$.

**Independent Set**
- **Input:** Graph $G$, integer $k$
- **Parameter:** $k$
- **Question:** Does $G$ have an independent set of size $k$?

- We know: **Independent Set** $\leq_p$ **Vertex Cover**
- However: **Vertex Cover** $\in$ **FPT** but **Independent Set** is not known to be in **FPT**
We will need another type of reductions

- Issue with polynomial-time reductions: parameter can change arbitrarily
We will need another type of reductions

- Issue with polynomial-time reductions: parameter can change arbitrarily
- We will want the reduction to produce an instance where the parameter is bounded by a function of the original instance
We will need another type of reductions

- Issue with polynomial-time reductions: parameter can change arbitrarily
- We will want the reduction to produce an instance where the parameter is bounded by a function of the original instance
- Also: we can allow the reduction to take FPT time instead of only polynomial time.
Outline

1. Reminder: Polynomial Time Reductions and NP-completeness

2. Parameterized Complexity Theory
   - Parameterized reductions
   - Parameterized complexity classes

3. Case studies

4. Further Reading
A **parameterized reduction** from a parameterized decision problem $\Pi_1$ to a parameterized decision problem $\Pi_2$ is an algorithm, which, for any instance $I$ of $\Pi_1$ with parameter $k$ produces an instance $I'$ of $\Pi_2$ with parameter $k'$ such that:

- $I$ is a \texttt{Yes}-instance for $\Pi_1$ $\iff$ $I'$ is a \texttt{Yes}-instance for $\Pi_2$,
- there exists a computable function $g$ such that $k' \leq g(k)$, and
- there exists a computable function $f$ such that the running time of the algorithm is $f(k) \cdot |I|^{O(1)}$.

If there exists a parameterized reduction from $\Pi_1$ to $\Pi_2$, we write $\Pi_1 \leq_{\text{FPT}} \Pi_2$.

**Note**: We can assume that $f$ and $g$ are non-decreasing.
Lemma 7

If $\Pi_1, \Pi_2$ are parameterized decision problems such that $\Pi_1 \leq_{\text{FPT}} \Pi_2$, then $\Pi_2 \in \text{FPT}$ implies $\Pi_1 \in \text{FPT}$.

Proof.

Exercise.
A Boolean formula in Conjunctive Normal Form (CNF) is a conjunction (AND) of disjunctions (OR) of literals (a Boolean variable or its negation).

A HORN formula is a CNF formula where each clause contains at most one positive literal.

For a CNF formula $F$ and an assignment $\tau : S \rightarrow \{0, 1\}$ to a subset $S$ of its variables, the formula $F[\tau]$ is obtained from $F$ by removing each clause that contains a literal that evaluates to 1 under $S$, and removing all literals that evaluate to 0 from the remaining clauses.

HORN-Backdoor Detection

Input: A CNF formula $F$ and an integer $k$.
Parameter: $k$
Question: Is there a subset $S$ of the variables of $F$ with $|S| \leq k$ such that for each assignment $\tau : S \rightarrow \{0, 1\}$, the formula $F[\tau]$ is a HORN formula?

Example: $(\neg a \lor b \lor c) \land (b \lor \neg c \lor \neg d) \land (a \lor b \lor \neg e) \land (\neg b \lor c \lor \neg e)$ with $k = 1$ is a Yes-instance, certified by $S = \{b\}$.

Show that HORN-Backdoor Detection is FPT using the fact that Vertex Cover is FPT.
Outline

1. Reminder: Polynomial Time Reductions and NP-completeness

2. Parameterized Complexity Theory
   - Parameterized reductions
   - Parameterized complexity classes

3. Case studies

4. Further Reading
A **Boolean circuit** is a directed acyclic graph with the nodes labeled as follows:

- every node of in-degree 0 is an **input node**,
- every node with in-degree 1 is a **negation node** ($\neg$), and
- every node with in-degree $\geq 2$ is either an **AND-node** ($\land$) or an **OR-node** ($\lor$).

Moreover, exactly one node with out-degree 0 is also labeled the **output node**. The **depth** of the circuit is the maximum length of a directed path from an input node to the output node. The **weft** of the circuit is the maximum number of nodes with in-degree $\geq 3$ on a directed path from an input node to the output node.
A depth-3, weft-1 Boolean circuit with inputs $a, b, c, d, e$. 
Given an assignment of Boolean values to the input gates, the circuit determines Boolean values at each node in the obvious way. If the value of the output node is 1 for an input assignment, we say that this assignment satisfies the circuit. The weight of an assignment is its number of 1s.

**Weighted Circuit Satisfiability (WCS)**

**Input:** A Boolean circuit $C$, an integer $k$

**Parameter:** $k$

**Question:** Is there an assignment with weight $k$ that satisfies $C$?

**Exercise:** Show that Weighted Circuit Satisfiability $\in$ XP.
WCS for special circuits

Definition 9
The class of circuits $C_{t,d}$ contains the circuits with weft $\leq t$ and depth $\leq d$.

For any class of circuits $C$, we can define the following problem.

**WCS[$C$]**

**Input:** A Boolean circuit $C \in C$, an integer $k$

**Parameter:** $k$

**Question:** Is there an assignment with weight $k$ that satisfies $C$?
Definition 10 (W-hierarchy)

Let $t \in \{1, 2, \ldots \}$. A parameterized problem $\Pi$ is in the parameterized complexity class $W[t]$ if there exists a parameterized reduction from $\Pi$ to $WCS[\mathcal{C}_{t,d}]$ for some constant $d \geq 1$. 
### Theorem 11

**Independent Set** \( \in W[1] \).

### Theorem 12

**Dominating Set** \( \in W[2] \).

**Recall**: A **dominating set** of a graph \( G = (V, E) \) is a set of vertices \( S \subseteq V \) such that \( N_G[S] = V \).

<table>
<thead>
<tr>
<th><strong>Dominating Set</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A graph ( G = (V, E) ) and an integer ( k )</td>
</tr>
<tr>
<td><strong>Parameter:</strong> ( k )</td>
</tr>
<tr>
<td><strong>Question:</strong> Does ( G ) have a dominating set of size at most ( k )?</td>
</tr>
</tbody>
</table>
Parameterized reductions from **Independent Set** to $\text{WCS}[C_{1,3}]$ and from **Dominating Set** to $\text{WCS}[C_{2,2}]$.

Setting an input node to 1 corresponds to adding the corresponding vertex to the independent set / dominating set.
**Definition 13**

Let $t \in \{1, 2, \ldots \}$. A parameterized decision problem $\Pi$ is $W[t]$-hard if for every parameterized decision problem $\Pi'$ in $W[t]$, there is a parameterized reduction from $\Pi'$ to $\Pi$. $\Pi$ is $W[t]$-complete if $\Pi \in W[t]$ and $\Pi$ is $W[t]$-hard.

It has been proved that **Independent Set** is $W[1]$-hard and **Dominating Set** is $W[2]$-hard. Therefore,

**Theorem 14**

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

**Theorem 15**

**Dominating Set** is $W[2]$-complete.
To show that a parameterized decision problem \( \Pi \) is \( W[t] \)-hard:

- Select a \( W[t] \)-hard problem \( \Pi' \)
- Show that \( \Pi' \leq_{\text{FPT}} \Pi \) by designing a parameterized reduction from \( \Pi' \) to \( \Pi \)
  - Design an algorithm, that, for any instance \( I' \) of \( \Pi' \) with parameter \( k' \), produces an equivalent instance \( I \) of \( \Pi \) with parameter \( k \)
  - Show that \( k \) is upper bounded by a function of \( k' \)
  - Show that there exists a function \( f \) such that the running time of the algorithm is \( f(k') \cdot |I'|^{O(1)} \)
Reminder: Polynomial Time Reductions and NP-completeness

Parameterized Complexity Theory
- Parameterized reductions
- Parameterized complexity classes

Case studies

Further Reading
A clique in a graph $G = (V, E)$ is a subset of its vertices $S \subseteq V$ such that every two vertices from $S$ are adjacent in $G$. 

**Clique**

Input: Graph $G = (V, E)$, integer $k$

Parameter: $k$

Question: Does $G$ have a clique of size $k$?

We will show that Clique is $W[1]$-hard by a parameterized reduction from Independent Set.
Clique is W[1]-hard

Lemma 16

\textbf{INDEPENDENT SET} \leq_{\textsc{FPT}} \textbf{CLIQUE}.

Proof.

Given any instance \((G = (V, E), k)\) for \textbf{INDEPENDENT SET}, we need to describe an \textsc{FPT} algorithm that constructs an equivalent instance \((G', k')\) for \textbf{CLIQUE} such that \(k' \leq g(k)\) for some computable function \(g\).
Clique is W[1]-hard

Lemma 16

**Independent Set \( \leq_{\text{FPT}} \text{Clique} \).**

Proof.

Given any instance \((G = (V, E), k)\) for **Independent Set**, we need to describe an **FPT** algorithm that constructs an equivalent instance \((G', k')\) for **Clique** such that \( k' \leq g(k) \) for some computable function \( g \).

**Construction.** Set \( k' \leftarrow k \) and \( G' \leftarrow \overline{G} = (V, \{uv : u, v \in V, u \neq v, uv \notin E\}) \).
Lemma 16

**Indpendent Set \( \leq_{\text{FPT}} \text{Clique} \).**

**Proof.**

Given any instance \((G = (V, E), k)\) for **Independent Set**, we need to describe an **FPT** algorithm that constructs an equivalent instance \((G', k')\) for **Clique** such that \(k' \leq g(k)\) for some computable function \(g\).

**Construction.** Set \(k' \leftarrow k\) and \(G' \leftarrow \overline{G} = (V, \{uv : u, v \in V, u \neq v, uv \notin E\})\).

**Equivalence.** We need to show that \((G, k)\) is a **Yes**-instance for **Independent Set** if and only if \((G', k')\) is a **Yes**-instance for **Clique**.
 Lemma 16

**INDEPENDENT SET \( \leq_{\text{FPT}} \) CLIQUE.**

**Proof.**

Given any instance \((G = (V, E), k)\) for **INDEPENDENT SET**, we need to describe an **FPT** algorithm that constructs an equivalent instance \((G', k')\) for **CLIQUE** such that \(k' \leq g(k)\) for some computable function \(g\).

**Construction.** Set \(k' \leftarrow k\) and \(G' \leftarrow \overline{G} = (V, \{uv : u, v \in V, u \neq v, uv \notin E\})\).

**Equivalence.** We need to show that \((G, k)\) is a **YES**-instance for **INDEPENDENT SET** if and only if \((G', k')\) is a **YES**-instance for **CLIQUE**.

\((\Rightarrow):\) Let \(S\) be an independent set of size \(k\) in \(G\). For every two vertices \(u, v \in S\), we have that \(uv \notin E\). Therefore, \(uv \in E(\overline{G})\) for every two vertices in \(S\). We conclude that \(S\) is a clique of size \(k\) in \(\overline{G}\).
Lemma 16

**Independent Set** $\leq_{\text{FPT}}$ **Clique**.

Proof.

Given any instance $(G = (V, E), k)$ for **Independent Set**, we need to describe an **FPT** algorithm that constructs an equivalent instance $(G', k')$ for **Clique** such that $k' \leq g(k)$ for some computable function $g$.

**Construction.** Set $k' \leftarrow k$ and $G' \leftarrow \overline{G} = (V, \{uv : u, v \in V, u \neq v, uv \notin E\})$.

**Equivalence.** We need to show that $(G, k)$ is a **Yes**-instance for **Independent Set** if and only if $(G', k')$ is a **Yes**-instance for **Clique**.

$(\Rightarrow)$: Let $S$ be an independent set of size $k$ in $G$. For every two vertices $u, v \in S$, we have that $uv \notin E$. Therefore, $uv \in E(\overline{G})$ for every two vertices in $S$. We conclude that $S$ is a clique of size $k$ in $\overline{G}$.

$(\Leftarrow)$: Let $S$ be a clique of size $k$ in $\overline{G}$. By a similar argument, $S$ is an independent set of size $k$ in $G$. 
Lemma 16

**Theorem.** \( \text{INDEPENDENT SET} \leq_{\text{FPT}} \text{CLIQUE} \).

**Proof.**

Given any instance \((G = (V, E), k)\) for \text{INDEPENDENT SET}, we need to describe an \text{FPT} algorithm that constructs an equivalent instance \((G', k')\) for \text{CLIQUE} such that \(k' \leq g(k)\) for some computable function \(g\).

**Construction.** Set \(k' \leftarrow k\) and \(G' \leftarrow \overline{G} = (V, \{uv : u, v \in V, u \neq v, uv \notin E\})\).

**Equivalence.** We need to show that \((G, k)\) is a \text{YES}-instance for \text{INDEPENDENT SET} if and only if \((G', k')\) is a \text{YES}-instance for \text{CLIQUE}.

\((\Rightarrow)\): Let \(S\) be an independent set of size \(k\) in \(G\). For every two vertices \(u, v \in S\), we have that \(uv \notin E\). Therefore, \(uv \in E(\overline{G})\) for every two vertices in \(S\). We conclude that \(S\) is a clique of size \(k\) in \(\overline{G}\).

\((\Leftarrow)\): Let \(S\) be a clique of size \(k\) in \(\overline{G}\). By a similar argument, \(S\) is an independent set of size \(k\) in \(G\).

**Parameter.** \(k' \leq k\).
Clique is \(W[1]\)-hard

**Lemma 16**

\(\text{INDEPENDENT SET} \leq_{FPT} \text{CLIQUE} \).

**Proof.**

Given any instance \((G = (V, E), k)\) for \text{INDEPENDENT SET}, we need to describe an \text{FPT} algorithm that constructs an equivalent instance \((G', k')\) for \text{CLIQUE} such that \(k' \leq g(k)\) for some computable function \(g\).

**Construction.** Set \(k' \leftarrow k\) and \(G' \leftarrow \overline{G} = (V, \{uv : u, v \in V, u \neq v, uv \notin E\})\).

**Equivalence.** We need to show that \((G, k)\) is a \text{Yes}-instance for \text{INDEPENDENT SET} if and only if \((G', k')\) is a \text{Yes}-instance for \text{CLIQUE}.

\((\Rightarrow)\): Let \(S\) be an independent set of size \(k\) in \(G\). For every two vertices \(u, v \in S\), we have that \(uv \notin E\). Therefore, \(uv \in E(\overline{G})\) for every two vertices in \(S\). We conclude that \(S\) is a clique of size \(k\) in \(\overline{G}\).

\((\Leftarrow)\): Let \(S\) be a clique of size \(k\) in \(\overline{G}\). By a similar argument, \(S\) is an independent set of size \(k\) in \(G\).

**Parameter.** \(k' \leq k\).

**Running time.** The construction can clearly be done in \text{FPT} time, and even in polynomial time.
Corollary 17

\textbf{Clique} \textit{is W[1]-hard}
**Recall:** A $k$-coloring of a graph $G = (V, E)$ is a function $f : V \rightarrow \{1, 2, \ldots, k\}$ assigning colors to $V$ such that no two adjacent vertices receive the same color.

**Multicolor Clique**

**Input:** A graph $G = (V, E)$, an integer $k$, and a $k$-coloring of $G$

**Parameter:** $k$

**Question:** Does $G$ have a clique of size $k$?

- Show that **Multicolor Clique** is $W[1]$-hard.
Recall: A $k$-coloring of a graph $G = (V, E)$ is a function $f : V \to \{1, 2, ..., k\}$ assigning colors to $V$ such that no two adjacent vertices receive the same color.

**Multicolor Clique**

Input: A graph $G = (V, E)$, an integer $k$, and a $k$-coloring of $G$

Parameter: $k$

Question: Does $G$ have a clique of size $k$?

- Show that **Multicolor Clique** is $W[1]$-hard.

**Hint:** Reduce from **Clique**, and create $k$ copies of $V$, each one being an independent set in $G'$. 

**Recall:** A \( k \)-coloring of a graph \( G = (V, E) \) is a function \( f : V \rightarrow \{1, 2, ..., k\} \) assigning colors to \( V \) such that no two adjacent vertices receive the same color.

**Multicolor Clique**

<table>
<thead>
<tr>
<th>Input:</th>
<th>A graph ( G = (V, E) ), an integer ( k ), and a ( k )-coloring of ( G )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter:</td>
<td>( k )</td>
</tr>
<tr>
<td>Question:</td>
<td>Does ( G ) have a clique of size ( k )?</td>
</tr>
</tbody>
</table>

- Show that **Multicolor Clique** is \( W[1] \)-hard.

**Hint:** Reduce from **Clique**, and create \( k \) copies of \( V \), each one being an independent set in \( G' \). Add edges to enforce constraints that a clique of size \( k \) in \( G' \) corresponds to a clique of size \( k \) in \( G \), and vice-versa.
Exercise

A set system $S$ is a pair $(V, H)$, where $V$ is a finite set of elements and $H$ is a set of subsets of $V$.

A set cover of a set system $S = (V, H)$ is a subset $X$ of $H$ such that each element of $V$ is contained in at least one of the sets in $X$, i.e., $\bigcup_{Y \in X} Y = V$.

\begin{center}
\textbf{Set Cover}
\end{center}

\begin{tabular}{ll}
Input: & A set system $S = (V, H)$ and an integer $k$ \\
Parameter: & $k$ \\
Question: & Does $S$ have a set cover of cardinality at most $k$? \\
\end{tabular}

Show that \textbf{Set Cover} is \textbf{W}[2]-hard.
Exercise

A set system $S$ is a pair $(V, H)$, where $V$ is a finite set of elements and $H$ is a set of subsets of $V$.

A set cover of a set system $S = (V, H)$ is a subset $X$ of $H$ such that each element of $V$ is contained in at least one of the sets in $X$, i.e., $\bigcup_{Y \in X} Y = V$.

**Set Cover**

Input: A set system $S = (V, H)$ and an integer $k$

Parameter: $k$

Question: Does $S$ have a set cover of cardinality at most $k$?

Show that Set Cover is $W[2]$-hard.

**Hint:** Reduce from Dominating Set.
A hitting set of a set system $S = (V, H)$ is a subset $X$ of $V$ such that $X$ contains at least one element of each set in $H$, i.e., $X \cap Y \neq \emptyset$ for each $Y \in H$.

**Hitting Set**

**Input:** A set system $S = (V, H)$ and an integer $k$

**Parameter:** $k$

**Question:** Does $S$ have a hitting set of size at most $k$?

Show that **Hitting Set** is $W[2]$-hard.
A hitting set of a set system \( S = (V, H) \) is a subset \( X \) of \( V \) such that \( X \) contains at least one element of each set in \( H \), i.e., \( X \cap Y \neq \emptyset \) for each \( Y \in H \).

**Hitting Set**

**Input:** A set system \( S = (V, H) \) and an integer \( k \)

**Parameter:** \( k \)

**Question:** Does \( S \) have a hitting set of size at most \( k \)?

![Diagram showing a hitting set]

- Show that **Hitting Set** is \( W[2] \)-hard.

**Hint:** Exploit a duality between sets and elements in set covers and hitting sets.
Outline

1. Reminder: Polynomial Time Reductions and NP-completeness

2. Parameterized Complexity Theory
   - Parameterized reductions
   - Parameterized complexity classes

3. Case studies

4. Further Reading
Further Reading

