1. Introduction

COMP6741: Parameterized and Exact Computation

Serge Gaspers$^{1,2}$

$^1$School of Computer Science and Engineering, UNSW Australia
$^2$Data61, Decision Sciences Group, CSIRO

Semester 2, 2016
Outline

1. Algorithms for NP-hard problems
2. Exponential Time Algorithms
3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover
4. Further Reading
Outline

1. Algorithms for NP-hard problems

2. Exponential Time Algorithms

3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover

4. Further Reading
Central question

\[ P \text{ vs. } NP \]
NP-hard problems

- no known polynomial time algorithm for any NP-hard problem
- belief: $P \neq NP$
- What to do when facing an NP-hard problem?
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 $S$.

**Example problem: Vertex Cover**

Input: Graph $G$, integer $k$

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

**Note:** **Vertex Cover** is **NP**-complete.
Coping with NP-hardness

- **Approximation algorithms**
  - There is an algorithm, which, given an instance \((G, k)\) for Vertex Cover, finds a vertex cover of size at most \(2k\) or correctly determines that \(G\) has no vertex cover of size \(k\).

- **Exact exponential time algorithms**
  - There is an algorithm solving Vertex Cover in time \(O(1.1970^n)\), where \(n = |V|\).

- **Fixed parameter algorithms**
  - There is an algorithm solving Vertex Cover in time \(O(1.2738^k + kn)\).

- **Heuristics**
  - The COVER heuristic (COVer Edges Randomly) finds a smaller vertex cover than state-of-the-art heuristics on a suite of hard benchmark instances.

- **Restricting the inputs**
  - Vertex Cover can be solved in polynomial time on bipartite graphs, trees, interval graphs, etc.

- **Quantum algorithms?**
  - Not believed to solve NP-hard problems in polynomial time.
Aims of this course

Design and analyze algorithms for \textbf{NP}-hard problems.

We focus on algorithms that solve \textbf{NP}-hard problems \textit{exactly} and analyze their worst case running time.
Outline

1. Algorithms for NP-hard problems

2. Exponential Time Algorithms

3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover

4. Further Reading
Running times

Worst case running time of an algorithm.

- An algorithm is **polynomial** if $\exists c \in \mathbb{N}$ such that the algorithm solves every instance in time $O(n^c)$, where $n$ is the size of the instance. Also: $n^{O(1)}$ or poly$(n)$.
- **quasi-polynomial**: $2^{O(\log^c n)}$, $c \in O(1)$
- **sub-exponential**: $2^{o(n)}$
- **exponential**: $2^{\text{poly}(n)}$
- **double-exponential**: $2^{2^{\text{poly}(n)}}$

$O^*$-notation ignores polynomial factors in the input size:

$$O^*(f(n)) \equiv O(f(n) \cdot \text{poly}(n))$$
$$O^*(f(k)) \equiv O(f(k) \cdot \text{poly}(n))$$
Brute-force algorithms for NP-hard problems

Theorem 1

Every problem in \text{NP} can be solved in exponential time.
Theorem 1

*Every problem in NP can be solved in exponential time.*

**Proof.**

Let \( \Pi \) be an arbitrary problem in NP. [Use certificate-based definition of NP]

We know that \( \exists \) a polynomial \( p \) and a polynomial-time verification algorithm \( V \) such that:

- for every \( x \in \Pi \) (i.e., every Yes-instance for \( \Pi \)) \( \exists \) string \( y \in \{0, 1\}^* \), \( |y| \leq p(|x|) \), such that \( V(x, y) = 1 \), and

- for every \( x \notin \Pi \) (i.e., every No-instance for \( \Pi \)) and every string \( y \in \{0, 1\}^* \), \( V(x, y) = 0 \).
Brute-force algorithms for NP-hard problems

Theorem 1

*Every problem in NP can be solved in exponential time.*

Proof.

Let $\Pi$ be an arbitrary problem in NP. [Use certificate-based definition of NP]

We know that $\exists$ a polynomial $p$ and a polynomial-time verification algorithm $V$ such that:

- for every $x \in \Pi$ (i.e., every Yes-instance for $\Pi$) $\exists$ string $y \in \{0, 1\}^*$, $|y| \leq p(|x|)$, such that $V(x, y) = 1$, and
- for every $x \notin \Pi$ (i.e., every No-instance for $\Pi$) and every string $y \in \{0, 1\}^*$, $V(x, y) = 0$.

Now, we can prove there exists an exponential-time algorithm for $\Pi$ with input $x$:

- For each string $y \in \{0, 1\}^*$ with $|y| \leq p(|x|)$, evaluate $V(x, y)$ and return Yes if $V(x, y) = 1$.
- Return No.

Running time: $2^{p(|x|)} \cdot n^{O(1)} \subseteq 2^{O(2 \cdot p(|x|))} = 2^{O(p(|x|))}$, but non-constructive. 

□
Three main categories for NP-complete problems

- Subset problems
- Permutation problems
- Partition problems
An independent set in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that the vertices in $S$ are pairwise non-adjacent in $G$.

**Independent Set**

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

Brute-force:
An independent set in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that the vertices in $S$ are pairwise non-adjacent in $G$.

**INDEPENDENT SET**

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

Brute-force: $O^*(2^n)$, where $n = |V(G)|$
Permutation Problem: Traveling Salesman

Traveling Salesman Problem (TSP)

Input: a set of $n$ cities, the distance $d(i, j) \in \mathbb{N}$ between every two cities $i$ and $j$, integer $k$

Question: Is there a permutation of the cities (a tour) such that the total distance when traveling from city to city in the specified order, and returning back to the origin, is at most $k$?

Brute-force:
**Permutation Problem: Traveling Salesman**

**Traveling Salesman Problem (TSP)**

**Input:** a set of \( n \) cities, the distance \( d(i, j) \in \mathbb{N} \) between every two cities \( i \) and \( j \), integer \( k \)

**Question:** Is there a permutation of the cities (a tour) such that the total distance when traveling from city to city in the specified order, and returning back to the origin, is at most \( k \)?

**Brute-force:** \( O^*(n!) \subseteq 2^{O(n \log n)} \)
Partition Problem: **COLORING**

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

**COLORING**

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

**Question:** Does $G$ have a $k$-coloring?

**Brute-force:**

$O^*(k^n)$, where $n = |V(G)|$
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.

**Coloring**

Input: Graph \( G \), integer \( k \)

Question: Does \( G \) have a \( k \)-coloring?

Brute-force: \( O^*(k^n) \), where \( n = |V(G)| \)
natural question in Algorithms:
design faster (worst-case analysis) algorithms for problems

might lead to practical algorithms
  for small instances
    you don’t want to design software where your client/boss can find with better solutions *by hand* than your software
  subroutines for
    (sub)exponential time approximation algorithms
    randomized algorithms with expected polynomial run time
exhaustive search
  trivial method
  try all candidate solutions (certificates) for a ground set on \( n \) elements
  running times for problems in NP
    - Subset Problems: \( O^*(2^n) \)
    - Permutation Problems: \( O^*(n!) \)
    - Partition Problems: \( O^*(e^n \log n) \)

faster exact algorithms
  for some problems, it is possible to obtain provably faster algorithms
  running times \( O(1.0836^n) \), \( O(1.4689^n) \), \( O(1.9977^n) \)
How large are the instances one can solve in practice?

<table>
<thead>
<tr>
<th>Available time</th>
<th>1 s  $2^{36}$</th>
<th>1 min  $2^{42}$</th>
<th>1 hour  $2^{48}$</th>
<th>3 days  $2^{54}$</th>
<th>6 months  $2^{60}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n^5$</td>
<td>147</td>
<td>337</td>
<td>776</td>
<td>1782</td>
<td>4096</td>
</tr>
<tr>
<td>$n^{10}$</td>
<td>12</td>
<td>18</td>
<td>27</td>
<td>42</td>
<td>64</td>
</tr>
<tr>
<td>$1.05^n$</td>
<td>511</td>
<td>596</td>
<td>681</td>
<td>767</td>
<td>852</td>
</tr>
<tr>
<td>$1.1^n$</td>
<td>261</td>
<td>305</td>
<td>349</td>
<td>392</td>
<td>436</td>
</tr>
<tr>
<td>$1.5^n$</td>
<td>61</td>
<td>71</td>
<td>82</td>
<td>92</td>
<td>102</td>
</tr>
<tr>
<td>$2^n$</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
<td>60</td>
</tr>
<tr>
<td>$5^n$</td>
<td>15</td>
<td>18</td>
<td>20</td>
<td>23</td>
<td>25</td>
</tr>
<tr>
<td>$n!$</td>
<td>13</td>
<td>15</td>
<td>16</td>
<td>18</td>
<td>19</td>
</tr>
</tbody>
</table>

Note: Intel Core i7 920 (Quad core) executes between $2^{36}$ and $2^{37}$ instructions per second at 2.66 GHz.
“For every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run.”

– Alan Perlis (1922-1990, programming languages, 1st recipient of Turing Award)
Suppose a $2^n$ algorithm enables us to solve instances up to size $x$

- Faster processors
  - processor speed doubles after 18–24 months (Moore’s law)
  - can solve instances up to size $x + 1$

- Faster algorithm
  - design an $O^*(2^{n/2}) \subseteq O(1.4143^n)$ time algorithm
  - can solve instances up to size $2 \cdot x$
Outline

1. Algorithms for NP-hard problems
2. Exponential Time Algorithms
3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover
4. Further Reading
A story

A computer scientist meets a biologist ...
Eliminating conflicts from experiments

\[ n = 1000 \text{ experiments,} \]
\[ k = 20 \text{ experiments failed} \]

<table>
<thead>
<tr>
<th>Theoretical</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2^n)</td>
<td>1.07 \cdot 10^{301}</td>
</tr>
<tr>
<td>(n^k)</td>
<td>10^{60}</td>
</tr>
<tr>
<td>(2^k \cdot n)</td>
<td>1.05 \cdot 10^{9}</td>
</tr>
</tbody>
</table>

Notes:
– We assume that \(2^{36}\) instructions are carried out per second.
– The Big Bang happened roughly \(13.5 \cdot 10^9\) years ago.
Goal of Parameterized Complexity

Confine the combinatorial explosion to a parameter $k$.

For which problem–parameter combinations can we find algorithms with running times of the form

$$f(k) \cdot n^{O(1)},$$

where the $f$ is a computable function independent of the input size $n$?
Examples of Parameters

A Parameterized Problem

Input: an instance of the problem
Parameter: a parameter $k$
Question: a **Yes/No** question about the instance and the parameter

- A parameter can be
  - input size (trivial parameterization)
  - solution size
  - related to the structure of the input (maximum degree, treewidth, branchwidth, genus, ...)
  - etc.
Main Complexity Classes

**P:** class of problems that can be solved in time $n^{O(1)}$

**FPT:** class of problems that can be solved in time $f(k) \cdot n^{O(1)}$

**W[·]:** parameterized intractability classes

**XP:** class of problems that can be solved in time $f(k) \cdot n^{g(k)}$

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

**Known:** If $\text{FPT} = W[1]$, then the Exponential Time Hypothesis fails, i.e. 3-SAT can be solved in time $2^{o(n)}$. 
Outline

1. Algorithms for NP-hard problems

2. Exponential Time Algorithms

3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover

4. Further Reading
Vertex Cover

**Vertex Cover (VC)**

**Input:** A graph $G = (V, E)$ on $n$ vertices, an integer $k$

**Parameter:** $k$

**Question:** Is there a set of vertices $C \subseteq V$ of size at most $k$ such that every edge has at least one endpoint in $C$?
Outline

1. Algorithms for NP-hard problems

2. Exponential Time Algorithms

3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover

4. Further Reading
Brute Force Algorithms

- $2^n \cdot n^{O(1)}$ not FPT
- $n^k \cdot n^{O(1)}$ not FPT
Algorithm \( \text{vc1}(G, k) \); 

1. \textbf{if} \( E = \emptyset \) \textbf{then}  
   \hspace{1cm} // all edges are covered  
   \hspace{1cm} \text{return} \ Yes 

2. \hspace{1cm} \text{return} \ Yes 

3. \textbf{else if} \( k = 0 \) \textbf{then}  
   \hspace{1cm} // we cannot select any vertex  
   \hspace{1cm} \text{return} \ No 

4. \hspace{1cm} \text{return} \ No 

5. \textbf{else}  

6. \hspace{1cm} \text{Select an edge} \ uv \in E; 

7. \hspace{1cm} \text{return} \ \text{vc1}(G - u, k - 1) \lor \text{vc1}(G - v, k - 1)
Let us look at an arbitrary execution of the algorithm. Recursive calls form a search tree $T$

- with depth $\leq k$
- where each node has $\leq 2$ children

$\Rightarrow T$ has $\leq 2^k$ leafs and $\leq 2^k - 1$ internal nodes

- at each node the algorithm spends time $n^{O(1)}$
- The running time is $O^*(2^k)$
A faster FPT Algorithm

Algorithm vc2 \((G, k)\):

1. If \(E = \emptyset\) then // all edges are covered
   2. Return Yes
3. Else if \(k = 0\) then // we used too many vertices
   4. Return No
5. Else if \(\Delta(G) \leq 2\) then // \(G\) has maximum degree \(\leq 2\)
   6. Solve the problem in polynomial time;
7. Else
   8. Select a vertex \(v\) of maximum degree;
   9. Return \(vc2(G - v, k - 1) \lor vc2(G - N[v], k - d(v))\)
Algorithm \( vc2(G, k) \);

1. if \( E = \emptyset \) then // all edges are covered
   2. return Yes
2. else if \( k = 0 \) then // we used too many vertices
   3. return No
3. else if \( \Delta(G) \leq 2 \) then // \( G \) has maximum degree \( \leq 2 \)
   4. Solve the problem in polynomial time;
5. else
   6. Select a vertex \( v \) of maximum degree;
   7. return \( vc2(G - v, k - 1) \lor vc2(G - N[v], k - d(v)) \)
Running time analysis of vc2

- Number of leafs of the search tree:

\[ T(k) \leq T(k - 1) + T(k - 3) \]
\[ x^k \leq x^{k-1} + x^{k-3} \]
\[ x^3 - x^2 - 1 \leq 0 \]

- The equation \( x^3 - x^2 - 1 = 0 \) has a unique positive real solution: \( x \approx 1.4655... \)

- Running time: \( 1.4656^k \cdot n^{O(1)} \)
Outline

1. Algorithms for NP-hard problems
2. Exponential Time Algorithms
3. Parameterized Complexity
   - FPT Algorithm for Vertex Cover
   - Algorithms for Vertex Cover
4. Further Reading
Further Reading

- **Exponential-time algorithms**

- **Parameterized Complexity**