

1. Introduction

COMP6741: Parameterized and Exact Computation

Serge Gaspers^{1,2}

¹School of Computer Science and Engineering, UNSW Sydney, Australia

²Decision Sciences Group, Data61, CSIRO, Australia

Semester 2, 2017

- 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

- 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 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?

Example problem: VERTEX COVER

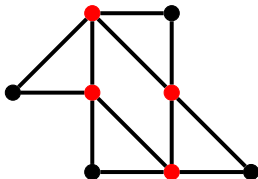
A **vertex cover** in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that every edge of G has an endpoint in S .

VERTEX COVER

Input: Graph G , integer k

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.

Design and analyze algorithms for NP-hard problems.

We focus on algorithms that solve NP-hard problems **exactly** and analyze their **worst case running time**.

- 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

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 $\text{poly}(n)$.

- **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:

$$O^*(f(n)) \equiv O(f(n) \cdot \text{poly}(n))$$

$$O^*(f(k)) \equiv O(f(k) \cdot \text{poly}(n))$$

Theorem 1

Every problem in NP can be solved in exponential time.

Brute-force algorithms for NP-hard problems

Theorem 1

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

Proof.

Let Π 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 Π) \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 Π) 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 Π 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 Π) \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 Π) and every string $y \in \{0, 1\}^*$, $V(x, y) = 0$.

Now, we can prove there exists an exponential-time algorithm for Π 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. \square

Three main categories for NP-complete problems

- Subset problems
- Permutation problems
- Partition problems

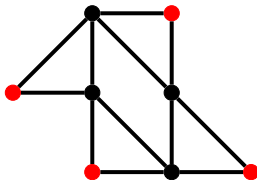
Subset Problem: INDEPENDENT SET

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:

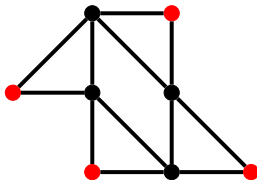
Subset Problem: INDEPENDENT SET

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 ?



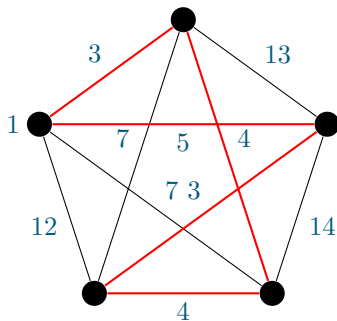
Brute-force: $O^*(2^n)$, where $n = |V(G)|$

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 ?



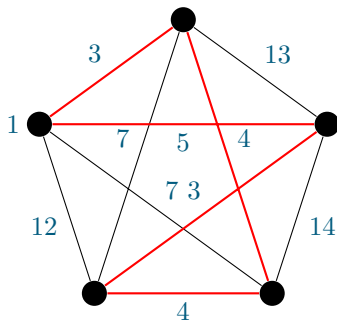
Brute-force:

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: $O^*(n!) \subseteq 2^{O(n \log n)}$

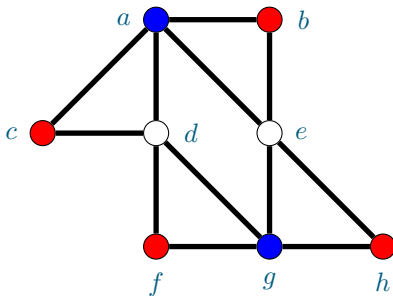
Partition Problem: COLORING

A k -coloring of a graph $G = (V, E)$ is a function $f : V \rightarrow \{1, 2, \dots, k\}$ 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:

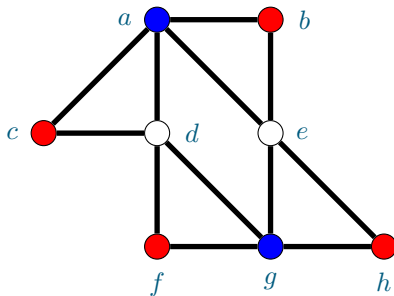
Partition Problem: COLORING

A k -coloring of a graph $G = (V, E)$ is a function $f : V \rightarrow \{1, 2, \dots, k\}$ 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 $n = |V(G)|$

Exponential Time Algorithms

- 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

Solve an NP-hard problem

- 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: $O^*(c^{n \log n})$
- faster exact algorithms
 - for some problems, it is possible to obtain provably faster algorithms
 - running times $O(1.0836^n)$, $O(1.4689^n)$, $O(1.9977^n)$

Exponential Time Algorithms in Practice

- How large are the instances one can solve in practice?

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

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)

Hardware vs. Algorithms

- 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 $2 \cdot x$

- 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

$n = 1000$ experiments,

$k = 20$ experiments failed

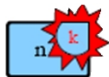
	Running Time	
Theoretical	Number of Instructions	Real
2^n	$1.07 \cdot 10^{301}$	$4.941 \cdot 10^{282}$ years
n^k	10^{60}	$4.611 \cdot 10^{41}$ years
$2^k \cdot n$	$1.05 \cdot 10^9$	0.01526 seconds

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 $f(k) \cdot n^{g(k)}$

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

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

- 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

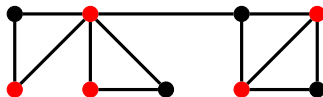
Vertex Cover

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 ?



- 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

An FPT Algorithm

Algorithm $\text{vc1}(G, k)$;

```
1 if  $E = \emptyset$  then // all edges are covered
2   return Yes
3 else if  $k \leq 0$  then // we cannot select any vertex
4   return No
5 else
6   Select an edge  $uv \in E$ ;
7   return  $\text{vc1}(G - u, k - 1) \vee \text{vc1}(G - v, k - 1)$ 
```

Running Time Analysis

- 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 ≤ 2 children
- $\Rightarrow T$ has $\leq 2^k$ leaves 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

A faster FPT Algorithm

Algorithm $\text{vc2}(G, k)$;

```
1 if  $E = \emptyset$  then // all edges are covered
2   return Yes
3 else if  $k \leq 0$  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  $\text{vc2}(G - v, k - 1) \vee \text{vc2}(G - N[v], k - d(v))$ 
```

- Number of leaves 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:
 $x \approx 1.4655\dots$
- Running time: $1.4656^k \cdot n^{O(1)}$

- 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

- Exponential-time algorithms

- Chapter 1, *Introduction* in Fedor V. Fomin and Dieter Kratsch. Exact Exponential Algorithms. Springer, 2010.
- Gerhard J. Woeginger: Exact Algorithms for NP-Hard Problems: A Survey. Combinatorial Optimization 2001: 185-208.
- Chapter 1, *Introduction* in Serge Gaspers. Exponential Time Algorithms: Structures, Measures, and Bounds. VDM Verlag Dr. Mueller, 2010.

- Parameterized Complexity

- Chapter 1, *Introduction* in Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Parameterized Algorithms. Springer, 2015.
- Chapter 2, *The Basic Definitions* in Rodney G. Downey and Michael R. Fellows. Fundamentals of Parameterized Complexity. Springer, 2013.
- Chapter I, *Foundations* in Rolf Niedermeier. Invitation to Fixed Parameter Algorithms. Oxford University Press, 2006.
- *Preface* in Jörg Flum and Martin Grohe. Parameterized Complexity Theory. Springer, 2006.