Parameterized Algorithms

Peter Rossmanith

Lehr- und Forschungsgebiet Theoretische Informatik
RWTH Aachen

October 15, 2021

Overview

Introduction

Parameterized algorithms are a method for the exact solution of hard problems.

Other such methods:

- Heuristics
- Simulated annealing
- Approximation algorithms
- Genetic algorithms
- Branch- and Bound
- Backtracking
- Total enumeration

NP-complete Problems

Many problems encountered in practice are NP-complete.

We know from complexity theory:

Definition

A language L is NP-complete, if

- $L \in NP$
- Every problem in NP can be reduced to L in polynomial time.

Theorem

If there is a polynomial time algorithm for an NP-complete problem, then $P = NP$.

Question: Does that mean that NP-complete problems are hard to solve in practice?

NP-complete problems

Why is SAT (satisfiability) NP-complete?

Because the computation of a nondeterministic Turing-maching can be simulated by a combinatorial circuit.

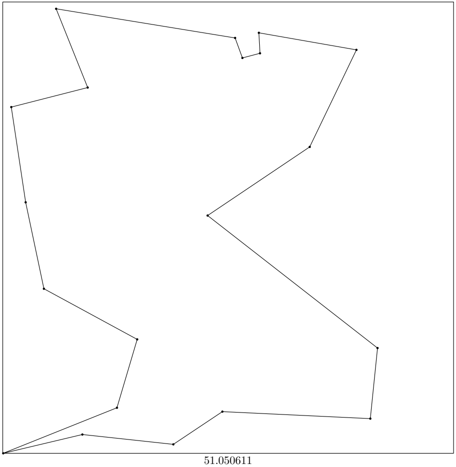
The existence of a successful computation of a Turingmachine can be reduced to the existence of a satisfying assignment for a circuit.

Therefore there are formulas whose satisfiability is as hard to determine as to solve any problem in *NP*.

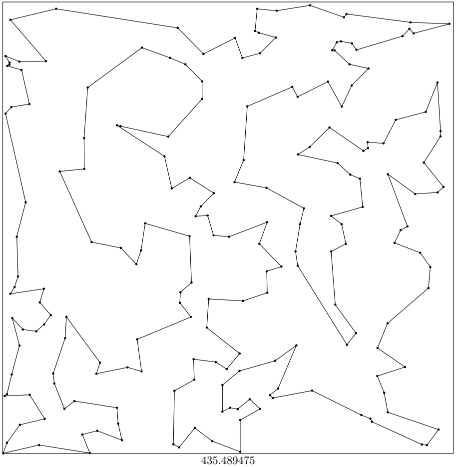
If look at the set of all formulas, then some of them are indeed very hard.

But most formulas are not constructed in this way!

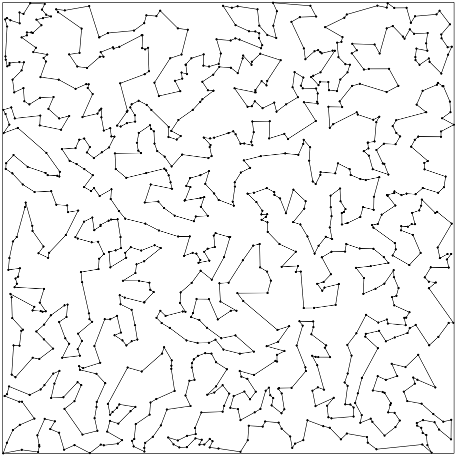
Example: TSP



Example: TSP



Example: TSP



2107.739054

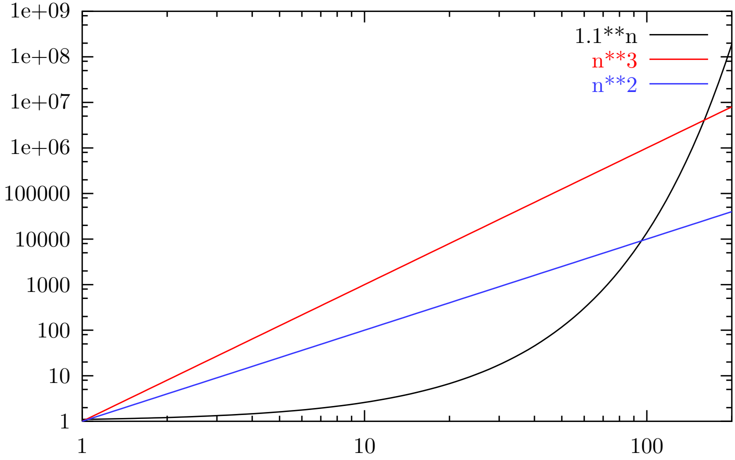
Running Times

NP-complete problems are hard in practice because there are no algorithms that always go in the right direction.

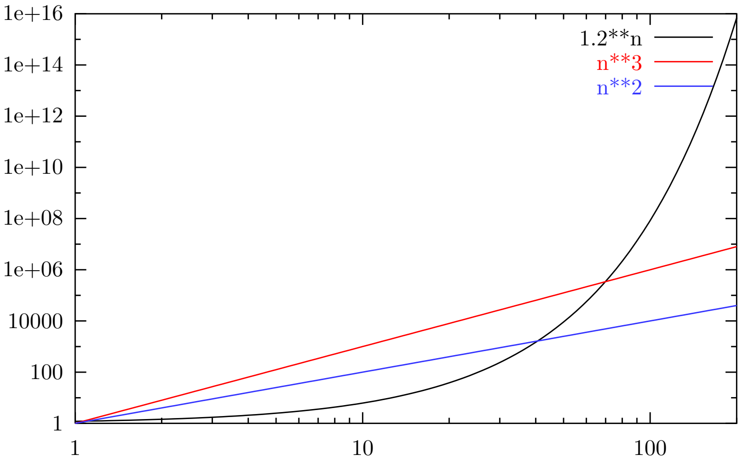
- Greedy-Algorithms
- Divide-and-Conquer
- Dynamic Programming

Hence, many wrong partial solutions have to be considered, leading to exponential running times.

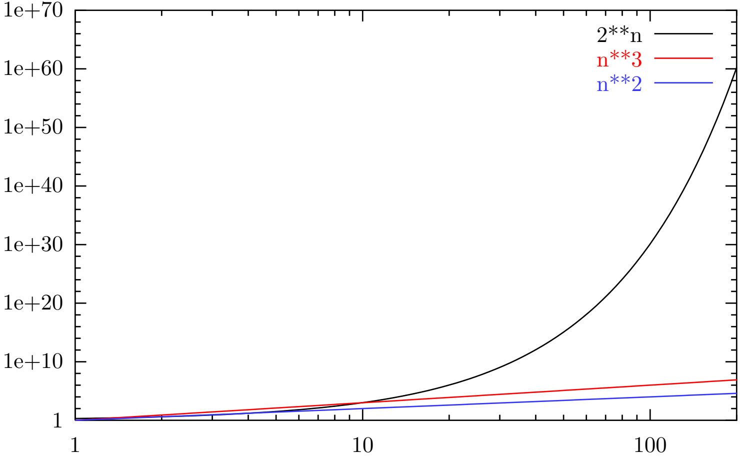
Comparing Running Times



Comparing Running Times



Comparing Running Times



NP-Completeness as an Excuse

Molecular biologist Joseph Felsenstein:

> *About ten years ago, some computer scientists came by and said they heard we have some really cool problems. They showed that the problems are NP-complete and went away!*

Overview

Easy and Hard Instances

- Exponential running time in the worst case
- Running time needs to be huge only for some instances
- Practical instances might be easy
- How can we distinguish between hard and easy instances?

Parameter

We assign a number, the parameter, to each instance.

Our hope:

- Good running times for small parameters
- Instances occuring in practice have small parameters

There is no contradiction to the NP-completeness of the problem!

Main Definition

Let there be an algorithmic problem.

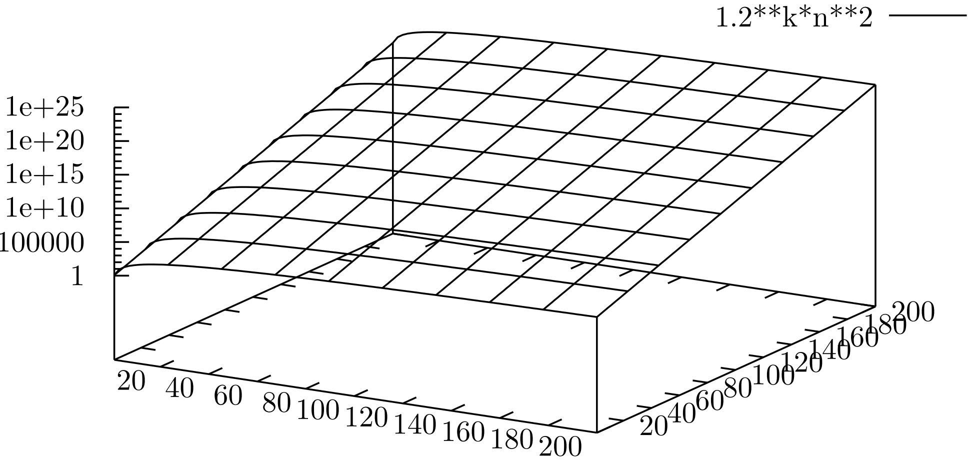
Let n be the size of some instance and k the corresponding parameter.

The problem is fixed parameter tractable, if there is an algorithm solving the problem whose running time is

$$O(f(k)n^c).$$

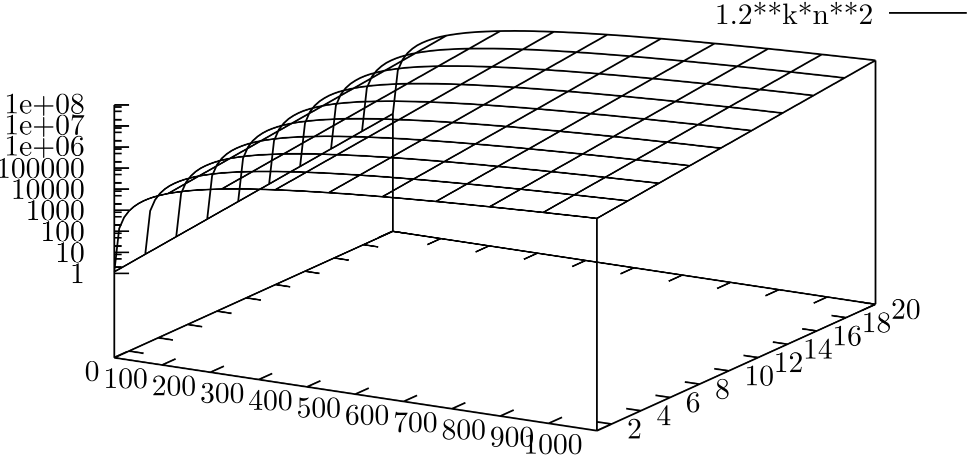
Here c is a constant and f an arbitrary function.

Running Time of a Parameterized Algorithm



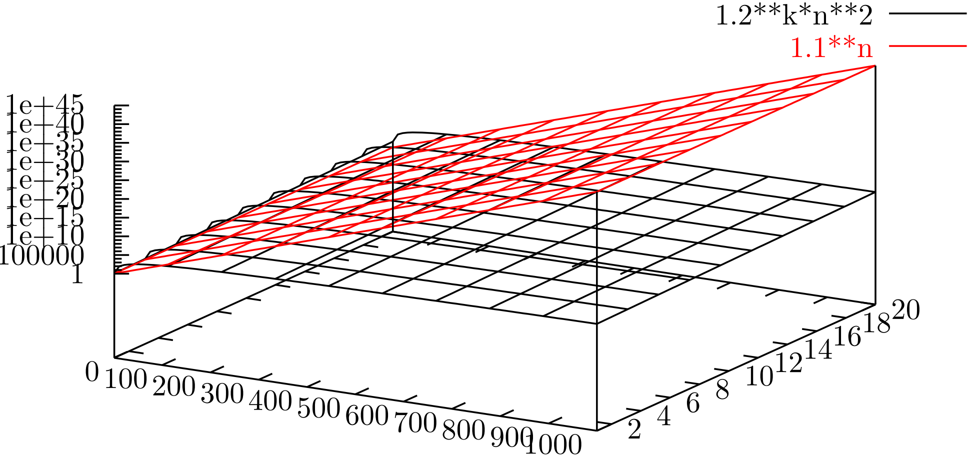
The running time is $1.2^k n^2$. The parameter is between 1 and n.

Running Time of a Parameterized Algorithm



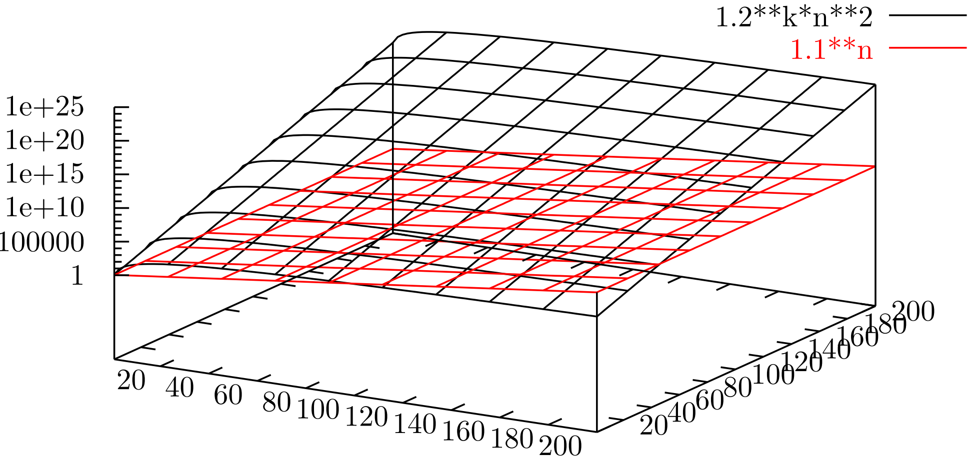
The running time is $1.2^k n^2$. The parameter is small.

Running Time of a Parameterized Algorithm



The running time is $1.2^k n^2$. The parameter is small. The non-parameterized algorithm has running time 1.1^n.

Running Time of a Parameterized Algorithm



Example: Vertex Cover

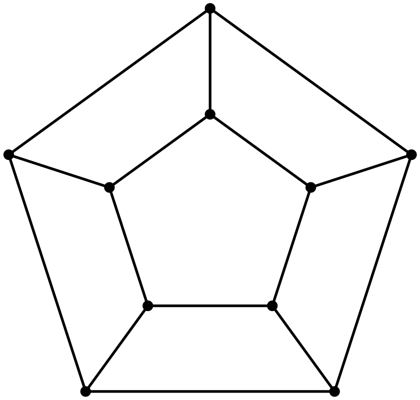
Input: A graph $G = (V, E)$.

Output: A minimal Vertex Cover $C \subseteq V$.

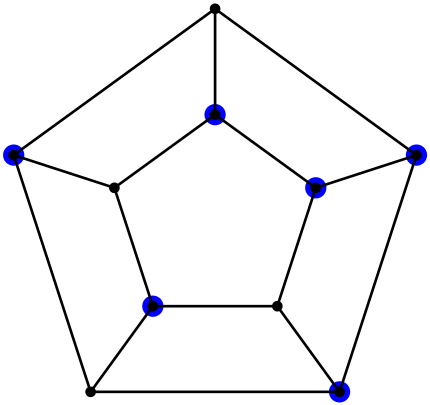
Definition

A set $C \subseteq V$ is a Vertex Cover of $G = (V, E)$, if at least one vertex of each edge in E is in C.

Example



Example



Expressing Vertex Cover as an ILP

Let $G = (V, E)$ be a graph with $V = \{v_1, \ldots, v_n\}$.

$$
\begin{aligned}
\text{Minimize } & v_1 + \ldots + v_n \\
\text{subject to } & 0 \leq v_i \leq 1 \text{ for } i = 1, \ldots, n \\
& v_i + v_j \geq 1 \text{ for } \{v_i, v_j\} \in E \\
& v_i \in \mathbf{Z} \text{ for } i = 1, \ldots, n
\end{aligned}
$$

Every NP-complete problem can be reduced to an ILP (but often it is a bad idea to do so).

British Museum Method

Many important NP-complete problems are indeed search problems. In some (very big) search space the solutions are well hidden.

One possible plan of attack is consequently to exhaustively search the whole search space.

In the case of vertex cover this amounts to looking at all $C \subseteq V$.

That makes $2^{|V|}$ different subsets.

The running time is $O(|E|2^{|V|})$.

Backtracking

Consider some vertex $v \in V$.

There are the two possibilities $v \in C$ or $v \notin C$.

If $v \notin C$, then $N(v) \subseteq C$, because all edges incident to v must be covered.

($N(v)$ is the neighborhood of v, i.e., all nodes adjacent to v.)

These simple observations lead immediately to an algorithm.

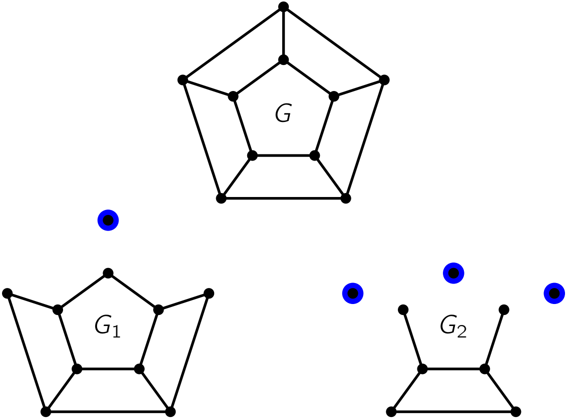
Backtracking

Input: $G = (V, E)$

Output: An optimal vertex cover $VC(G)$

if $V = \emptyset$ **then return** $\emptyset$
Choose an arbitrary node $v \in G$
$G_1 := (V - \{v\}, \{ e \in E \mid v \notin e \})$
$G_2 := (V - \{v\} - N(v), \{ e \in E \mid e \cap N(v) = \emptyset \})$
if $|\{v\} \cup VC(G_1)| \leq |N(v) \cup VC(G_2)|$
then return $\{v\} \cup VC(G_1)$
else return $N(v) \cup VC(G_2)$

Backtracking



Backtracking (a different approach)

Every edge $e = \{v_1, v_2\}$ must be covered by v_1 of v_2.

Hence, we can look at an edge $\{v_1, v_2\}$ and try recursively both possibilities:

- $v_1 \in C$
- $v_2 \in C$

This again leads to immeadiately to a simple algorithm:

Backtracking (a different approach)

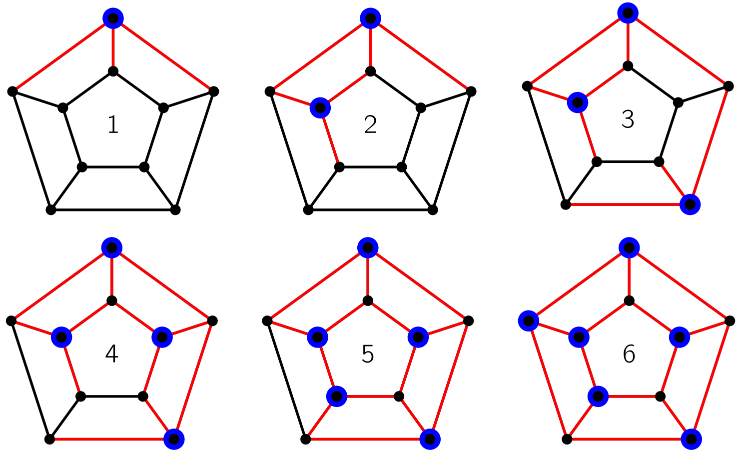
Input: $G = (V, E)$

Output: An optimal vertex cover $VC(G)$

if $E = \emptyset$ **then return** $\emptyset$
Choose some edge $\{v_1, v_2\} \in E$
$G_1 := (V - \{v_1\}, \{\, e \in E \mid v_1 \notin e \,\})$
$G_2 := (V - \{v_2\}, \{\, e \in E \mid v_2 \notin e \,\})$
if $|\{v_1\} \cup VC(G_1)| \leq |\{v_2\} \cup VC(G_2)|$
then return $\{v_1\} \cup VC(G_1)$
else return $\{v_2\} \cup VC(G_2)$

This recursive algorithms computes an optimal vertex cover.

Heuristics



Always choose a vertex with maximal degree (greedy).

Approximation Algorithms

Every edge has to be covered by at least one of its vertices.

Problem: Which one?

Solution: Take both.

- Naturally there is no guarantee that we find an optimal solution.
- The vertex cover found in this way can be at most twice as big as an optimal one.

Approximation algorithms

The algorithm might look like this:

```
C := ∅;
while E ≠ ∅ do
   Choose some e ∈ E;
   V := V − e;
   C := C ∪ e;
   E := { e' ∈ E | e ∩ e' = ∅ }
od;
return C
```

Parameterized Algorithm

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

Parameter: k

Output: A vertex cover $VC(G, k)$ of size k or smaller, if it exists.

if $E = \emptyset$ **then return** $\emptyset$
if $k = 0$ **then return** "no solution"
Choose some edge $\{v_1, v_2\} \in E$
$G_1 := (V - \{v_1\}, \{\, e \in E \mid v_1 \notin e \,\})$
$G_2 := (V - \{v_2\}, \{\, e \in E \mid v_2 \notin e \,\})$
if $|\{v_1\} \cup VC(G_1, k-1)| \leq |\{v_2\} \cup VC(G_2, k-1)|$
then return $\{v_1\} \cup VC(G_1, k-1)$
else return $\{v_2\} \cup VC(G_2, k-1)$