SAT: Introduction

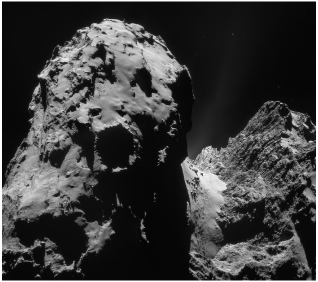
Mohamed Siala
https://siala.github.io

INSA-Toulouse & LAAS-CNRS

January 13, 2022

Context: Solving (Very) Hard Combinatorial Problems

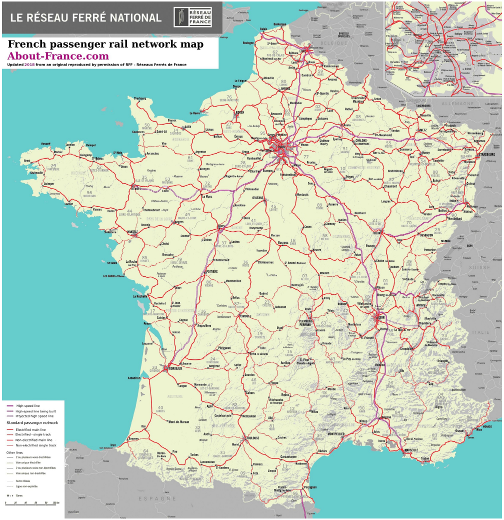
Context: Solving (Very) Hard Combinatorial Problems



https://homepages.laas.fr/ehebrard/rosetta.html

Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems



Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems

Context: Solving (Very) Hard Combinatorial Problems

Why this Lecture?

- I noticed that most graduate students are doing software development.

Why this Lecture?

- I noticed that most graduate students are doing software development.
- We are missing job opportunities in optimisation!

Why this Lecture?

- I noticed that most graduate students are doing software development.
- We are missing job opportunities in optimisation!
- Resources: many.. a good start would be the online course on discrete optimisation `https://www.coursera.org/learn/discrete-optimization`

Solving Methodologies

Solving Methodologies

1. Adhoc methods

Solving Methodologies

1. Adhoc methods

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm
    2. Heuristic method

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approached

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm
    2. Heuristic method
    3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approached
    1. (Mixed) Integer Programming,

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm
    2. Heuristic method
    3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approached
    1. (Mixed) Integer Programming,
    2. Constraint Programming

Solving Methodologies

1. Adhoc methods
    1. Specific exact algorithm
    2. Heuristic method
    3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approached
    1. (Mixed) Integer Programming,
    2. Constraint Programming
    3. Boolean Satisfiability (SAT)
    4. ...

Why Declarative Approaches?

Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approached
   1. (Mixed) Integer Programming,
   2. Constraint Programming
   3. Boolean Satisfiability (SAT)
   4. ...

Why Declarative Approaches?

- They are problem independent! The user models the problem in a specific language and the solver do the job!

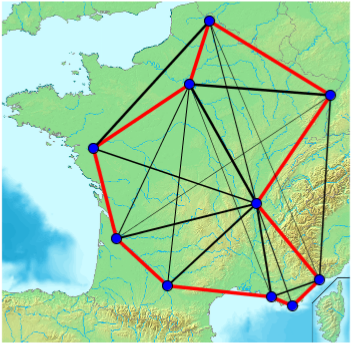
Solving Methodologies

1. Adhoc methods
   1. Specific exact algorithm
   2. Heuristic method
   3. Meta-heuristic (genetic algorithms, ant colony, ..)
2. Declarative Approached
   1. (Mixed) Integer Programming,
   2. Constraint Programming
   3. Boolean Satisfiability (SAT)
   4. ...

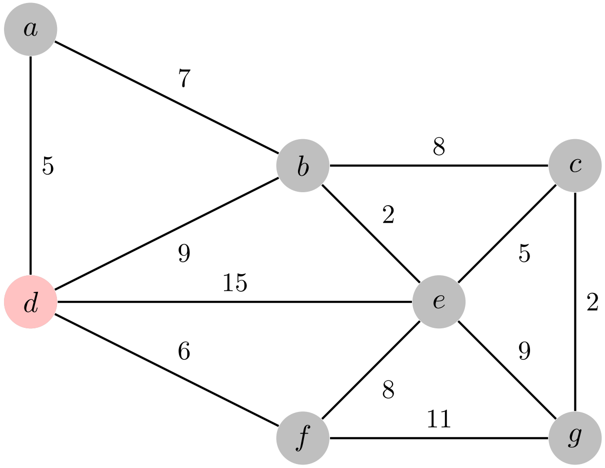
Why Declarative Approaches?

- They are problem independent! The user models the problem in a specific language and the solver do the job!
- Very active community

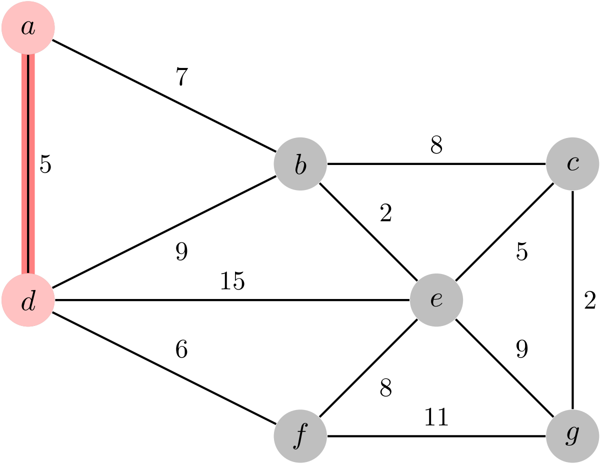
Travelling Salesman Problem



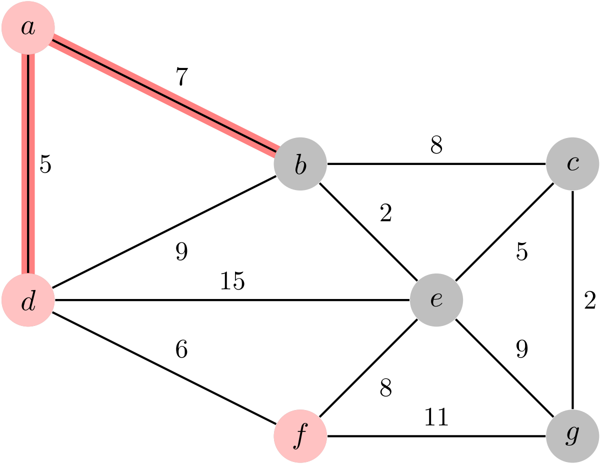
Exemple



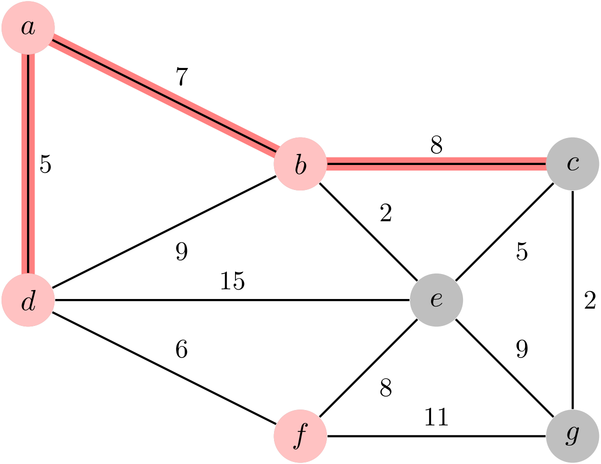
Exemple



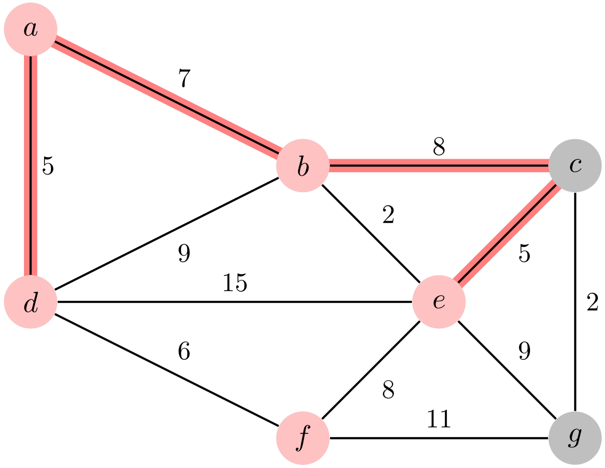
Exemple



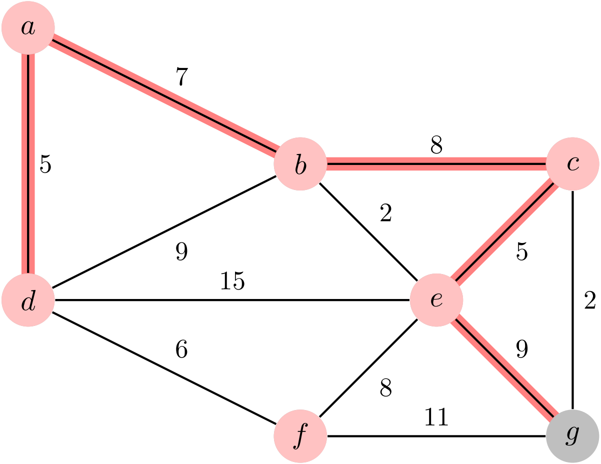
Exemple



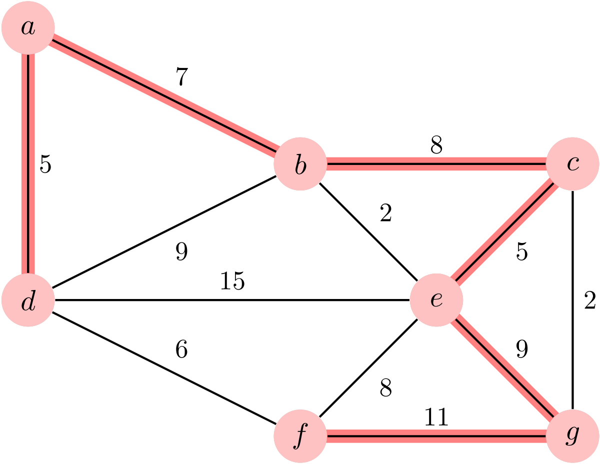
Exemple



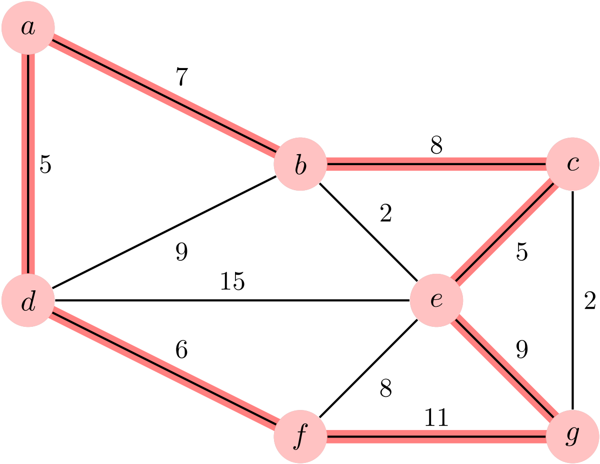
Exemple



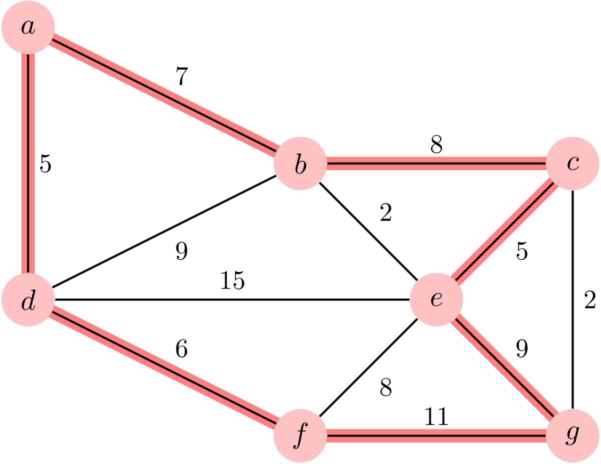
Exemple



Exemple

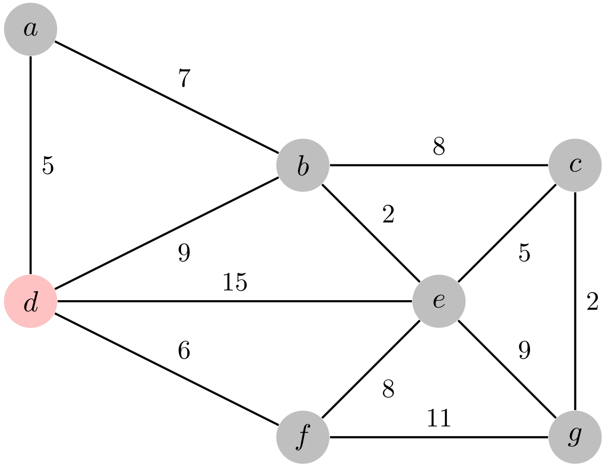


Exemple

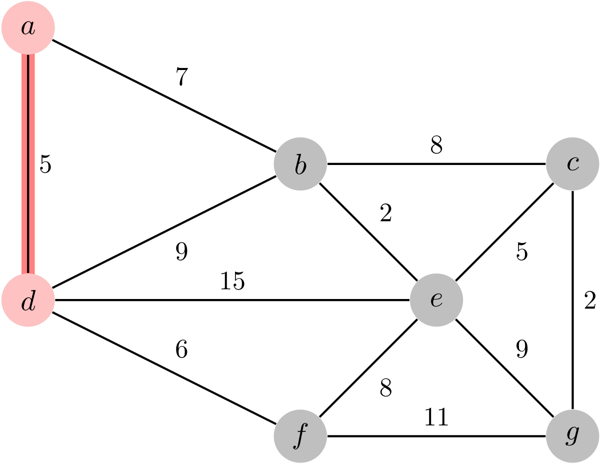


$--> Cost : 5 + 7 + 8 + 5 + 9 + 11 + 6 = 53Km$

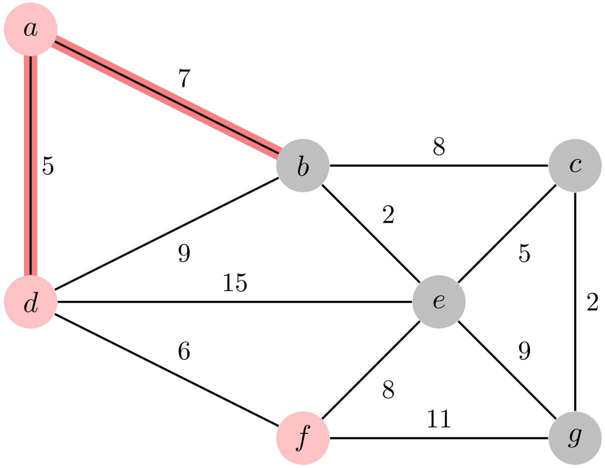
Example



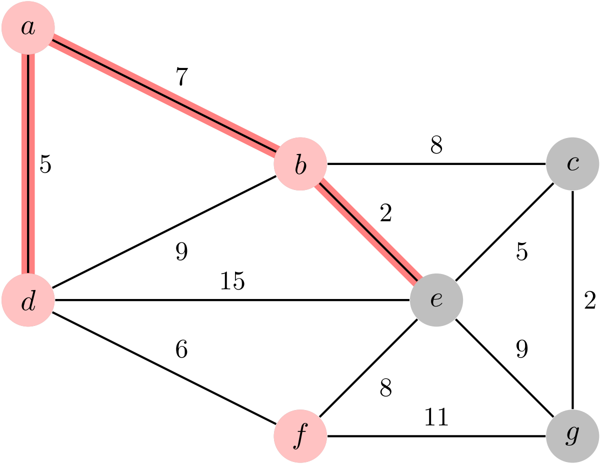
Example



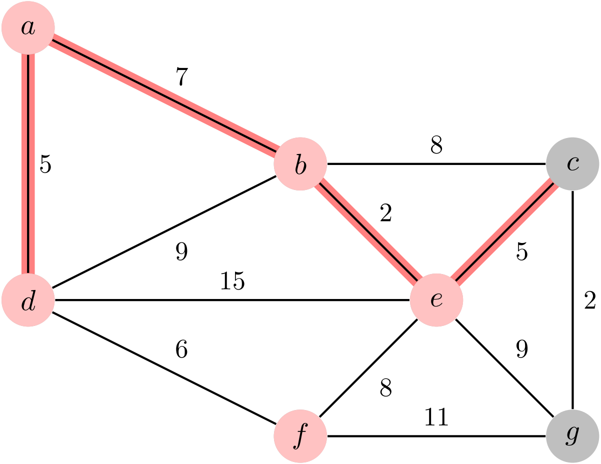
Example



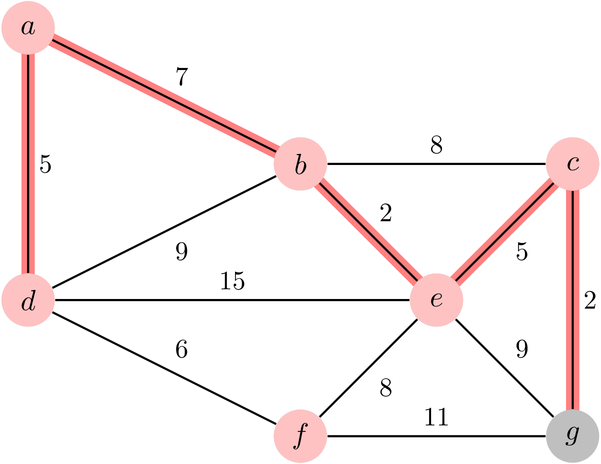
Example



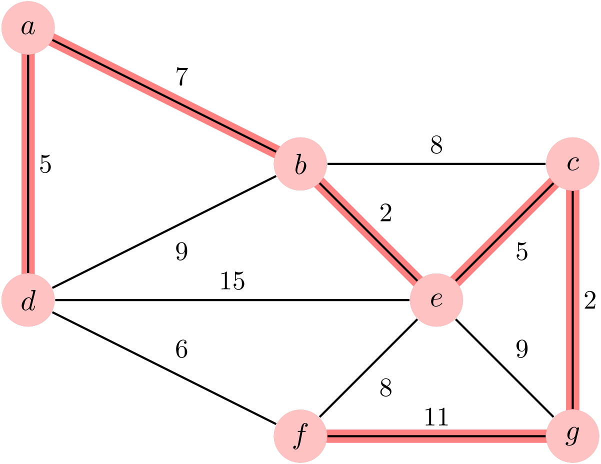
Example



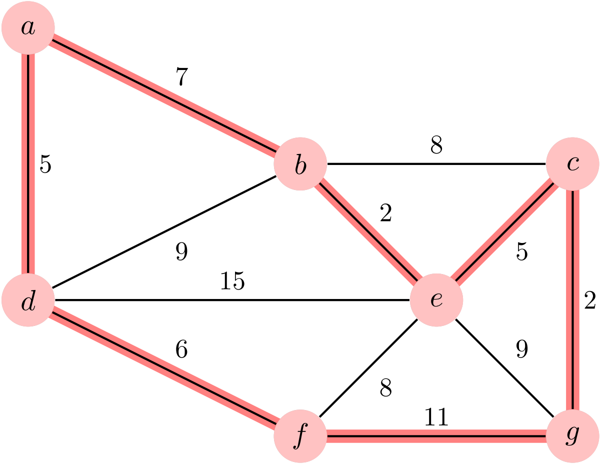
Example



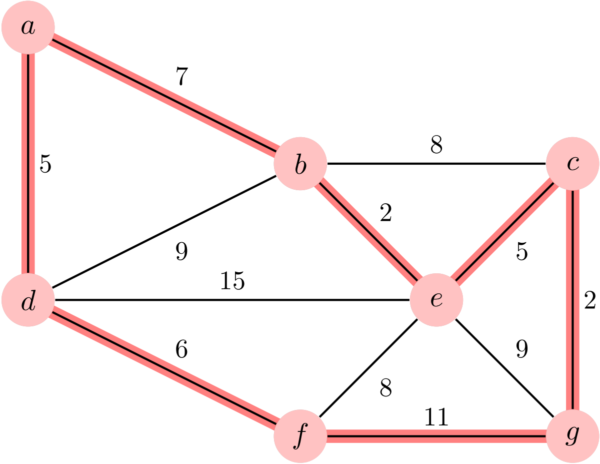
Example



Example



Example



$$--> Cost : 5 + 7 + 2 + 5 + 2 + 11 + 6 = 38Km$$

What if we check all possibilities?

What if we check all possibilities?

- 2 Cities $\rightarrow$1

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities

What if we check all possibilities?

- 2 Cities →1
- 5 Cities →24
- 8 Cities →4032
- 40 Cities →2.10^{46} (with a modern machine: 3.10^{27} years!)

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities $\rightarrow 2.10^{46}$ (with a modern machine: 3.10^{27} years!)
- 95 Cities, if we use a Plack (the shortest possible time interval that can be measured) processor and fill the universe with a processor per mm^3, we need $3\times$ the age of the universe

What if we check all possibilities?

- 2 Cities $\rightarrow$1
- 5 Cities $\rightarrow$24
- 8 Cities $\rightarrow$4032
- 40 Cities $\rightarrow 2.10^{46}$ (with a modern machine: 3.10^{27} years!)
- 95 Cities, if we use a Plack (the shortest possible time interval that can be measured) processor and fill the universe with a processor per mm^3, we need $3\times$ the age of the universe

The problem is inherently hard. However, the Concorde algorithm can solve instances up to 86 000 cities!

A step back: Problems, Instances, and Algorithms

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input of an output

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input of an output
- Many instances (instantiation of the input) for the same problem

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input of an output
- Many instances (instantiation of the input) for the same problem
- Many algorithms (methodologies) to solve the same problem

A step back: Problems, Instances, and Algorithms

- A problem is a question that associates an input of an output
- Many instances (instantiation of the input) for the same problem
- Many algorithms (methodologies) to solve the same problem

Example: The Sorting Integers problem

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7
  - 2, 3, 9, 8, 7

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7
  - 2, 3, 9, 8, 7
  - 2, 3, 7, 9, 8

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7
  - 2, 3, 9, 8, 7
  - 2, 3, 7, 9, 8
  - 2, 3, 7, 8, 9

Example: The Sorting Integers problem

- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7
  - 2, 3, 9, 8, 7
  - 2, 3, 7, 9, 8
  - 2, 3, 7, 8, 9
  - 2, 3, 7, 8, 9

Complexity

Complexity

- Complexity: a measure to analyze/classify algorithms based on the amount of resource required (Time and Memory)

Complexity

- Complexity: a measure to analyze/classify algorithms based on the amount of resource required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input

Complexity

- Complexity: a measure to analyze/classify algorithms based on the amount of resource required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input

Complexity

- Complexity: a measure to analyze/classify algorithms based on the amount of resource required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input
- The evaluation is made usually by reasoning about the worst case.

Complexity

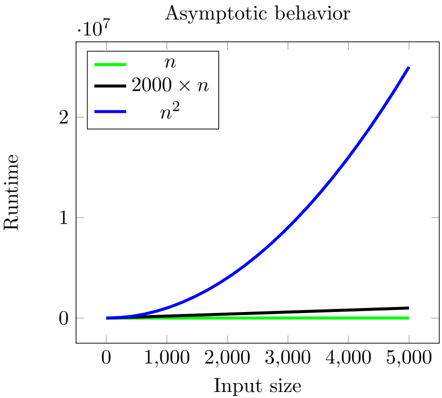
- Complexity: a measure to analyze/classify algorithms based on the amount of resource required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input
- The evaluation is made usually by reasoning about the worst case.
- The analysis is given with regard with the asymptotic behaviour

Complexity

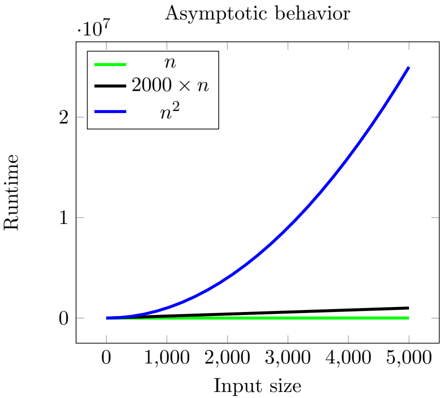
- Complexity: a measure to analyze/classify algorithms based on the amount of resource required (Time and Memory)
- Time Complexity: number of operations as a function of the size of the input
- Space Complexity: memory occupied by the algorithm as a function of the size of the input
- The evaluation is made usually by reasoning about the worst case.
- The analysis is given with regard with the asymptotic behaviour

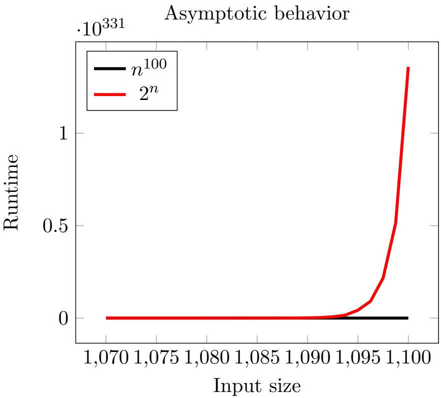
Asymptotic behaviour

Asymptotic behaviour



Asymptotic behaviour





- If f is a polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$

- If f is a polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$
- Convention:
  - Easy/Tractable Problem: We know a polynomial time algorithm to solve the problem
  - Hard/Intractable: No known polynomial algorithm

- If f is a polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$
- Convention:
  - Easy/Tractable Problem: We know a polynomial time algorithm to solve the problem
  - Hard/Intractable: No known polynomial algorithm
- Example: Th sorting problem is easy because we have an algorithm that runs in the worst case in $O(n^2)$ (and actually the same for memory consumption)

- If f is a polynomial and g is exponential then $f \in O(g)$.
  For instance $n^{10000} \in O(2^n)$
- Convention:
  - Easy/Tractable Problem: We know a polynomial time algorithm to solve the problem
  - Hard/Intractable: No known polynomial algorithm
- Example: Th sorting problem is easy because we have an algorithm that runs in the worst case in $O(n^2)$ (and actually the same for memory consumption)
- What if we don't know if a problem has a polynomial time algorithm?

Classes of problems

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm
- We know that $P \in NP$ (if you can solve then you can verify)

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm
- We know that $P \in NP$ (if you can solve then you can verify)
- For many Problems in NP, we don't know if a polynomial time algorithm exists.

Classes of problems

- **P** is the class of problems that are **solvable** in polynomial time (easy problems)
- **NP** is the class of problems that are **verifiable** in polynomial time algorithm
- We know that $P \in NP$ (if you can solve then you can verify)
- For many Problems in NP, we don't know if a polynomial time algorithm exists.
- **1 Million $** question: Is P=NP?

The Boolean Satisfiability Problem (SAT)

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$
- Propositional formula Φ given in a `Conjunctive Normal Form` (`CNF`) $\Phi : c_1 \wedge .. \wedge c_n$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$
- Propositional formula Φ given in a `Conjunctive Normal Form` (`CNF`) $\Phi : c_1 \wedge .. \wedge c_n$

The Boolean Satisfiability Problem (SAT)

Definitions

- Atoms (Boolean variables): x_1, x_2, ...
- Literal: $x_1, \neg x_1$
- Clauses: a clause is a disjunction of literals
- Example of clause: $(\neg x_1 \vee \neg x_4 \vee x_7)$
- Propositional formula Φ given in a `Conjunctive Normal Form` (`CNF`) $\Phi : c_1 \wedge .. \wedge c_n$

Given a set of Boolean variables $x_1, \ldots x_n$ and a CNF formulae Φ over $x_1, \ldots x_n$, the Boolean Satisfiability problem (SAT) is to find an assignment of the variables that satisfies all the clauses.

Why SAT?

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)
- Many theoretical properties

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)
- Many theoretical properties
- Huge practical improvements in the past 2 decades

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)
- Many theoretical properties
- Huge practical improvements in the past 2 decades
- Is considered today as a powerful technology to solve computational problems

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)
- Many theoretical properties
- Huge practical improvements in the past 2 decades
- Is considered today as a powerful technology to solve computational problems

In this lecture, we focus on the practical side

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)
- Many theoretical properties
- Huge practical improvements in the past 2 decades
- Is considered today as a powerful technology to solve computational problems

In this lecture, we focus on the practical side

- How to use it to solve problems (Modelling)

Why SAT?

- SAT is the first problem that is shown to be in the class NP-Complete (the hardest problems in NP)
- Many theoretical properties
- Huge practical improvements in the past 2 decades
- Is considered today as a powerful technology to solve computational problems

In this lecture, we focus on the practical side

- How to use it to solve problems (Modelling)
- Discover some efficient implementations

Example

Example

$x \vee \neg y \vee z$

$\neg x \vee \neg z$

$y \vee w$

$\neg w \vee \neg x$

Example

$x \vee \neg y \vee z$

$\neg x \vee \neg z$

$y \vee w$

$\neg w \vee \neg x$

A possible solution:

$$x \leftarrow 1; y \leftarrow 1; z \leftarrow 0; w \leftarrow 0$$

Modelling in SAT: The example of Graph Coloring

Modelling in SAT: The example of Graph Coloring

Graph Coloring is a well know combinatorial problem that has many applications (in particular in scheduling problems).

Modelling in SAT: The example of Graph Coloring

Graph Coloring is a well know combinatorial problem that has many applications (in particular in scheduling problems).

Let $G = (V, E)$ be an undirected graph where V is a set of n vertices and E is a set of m edges. Is it possible to colour the graph with k colours such that no two adjacent nodes share the same colour?

Modelling in SAT: The example of Graph Coloring

Graph Coloring is a well know combinatorial problem that has many applications (in particular in scheduling problems).

Let $G = (V, E)$ be an undirected graph where V is a set of n vertices and E is a set of m edges. Is it possible to colour the graph with k colours such that no two adjacent nodes share the same colour?