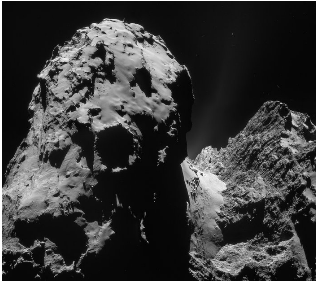
SAT: Modelling and Implementations

Mohamed Siala
https://siala.github.io

INSA-Toulouse & LAAS-CNRS

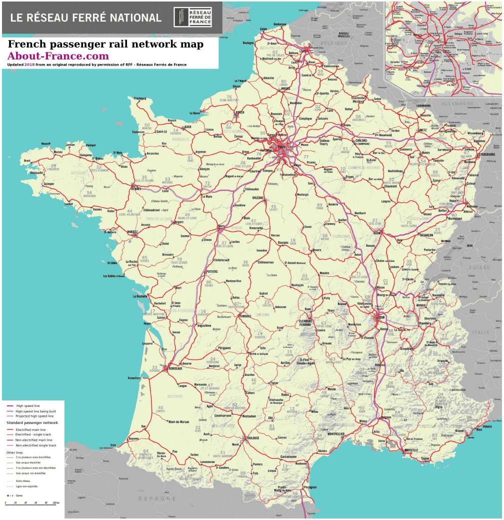
January 7, 2022

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

Why this Lecture?

- What I noticed that most graduate students are doing software development.
- We are missing job opportunities in optimisation!
- Start with the online course on optimisation https://www.coursera.org/learn/discrete-optimization by Pascal Van Hentenryck

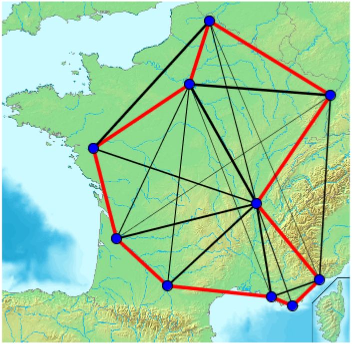
Solving Methodologies

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. ...

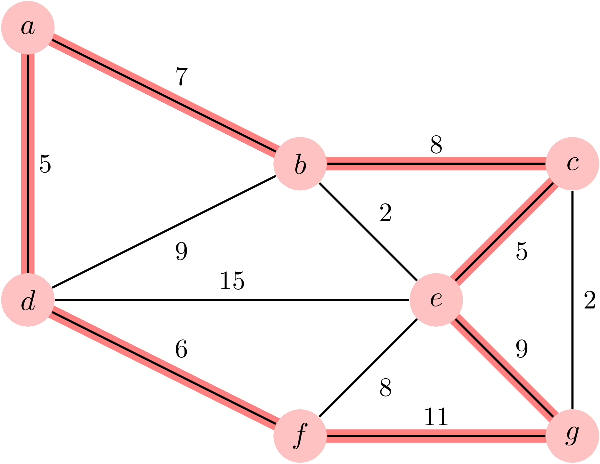
Why Declarative Approaches?

- 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

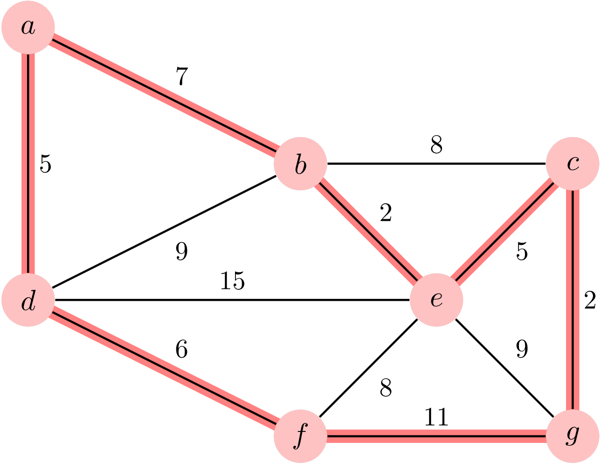


Exemple



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

Example



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

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 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

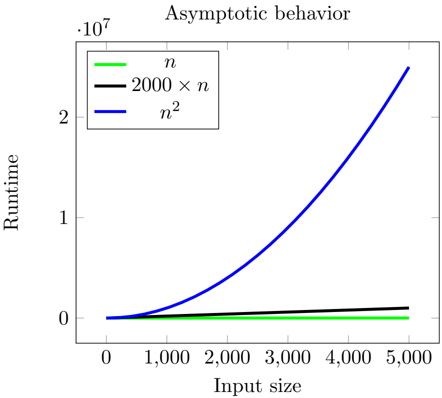
Example: The Sorting Integers problem

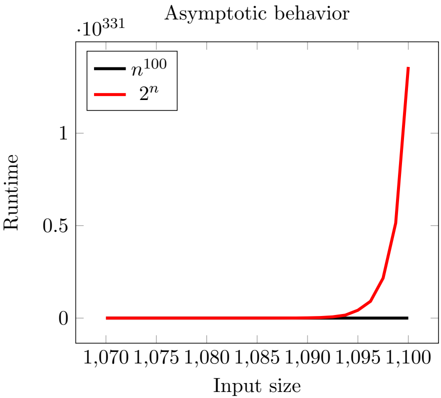
- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7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

Complexity

- Complexity: 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





- 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

- **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)

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?

- 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

$x \vee \neg y \vee z$

$\neg x \vee \neg z$

$y \vee w$

$\neg w \vee \neg x$

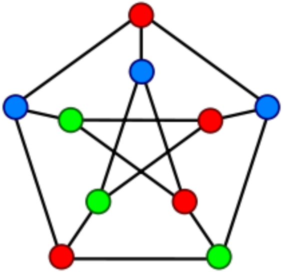
A possible solution:

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

Modelling in SAT: The example of Graph Coloring

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 the set of vertices and E is the set of 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

- Propose a SAT model for this problem (hint $x \rightarrow y$ is equivalent to $\neg x \vee y$).
- Propose a method that uses SAT for the minimisation version of the problem? That is, given $G = (V, E)$, we seek to find the minimum value of k to satisfy the colouring requirements.