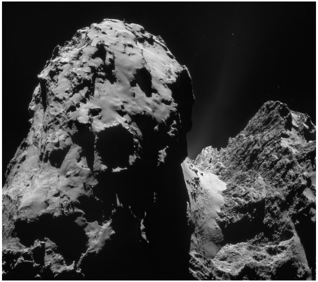
SAT: Modelling and Implementations

Mohamed Siala
https://siala.github.io

INSA-Toulouse & LAAS-CNRS

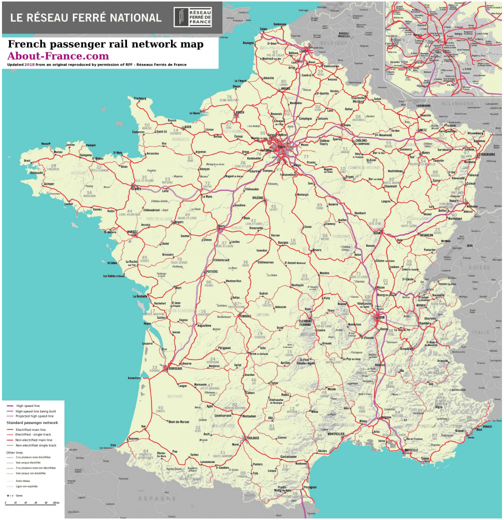
January 10, 2022

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?

- 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`

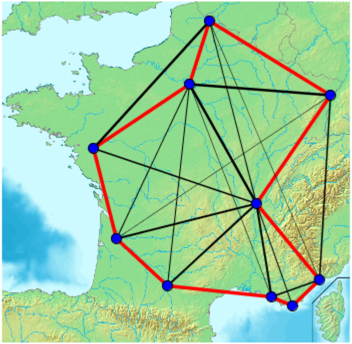
Solving Methodologies

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. ...

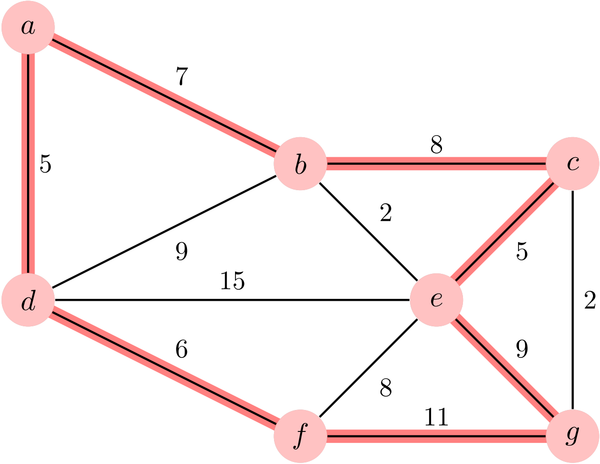
Why Declarative Approaches?

- 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

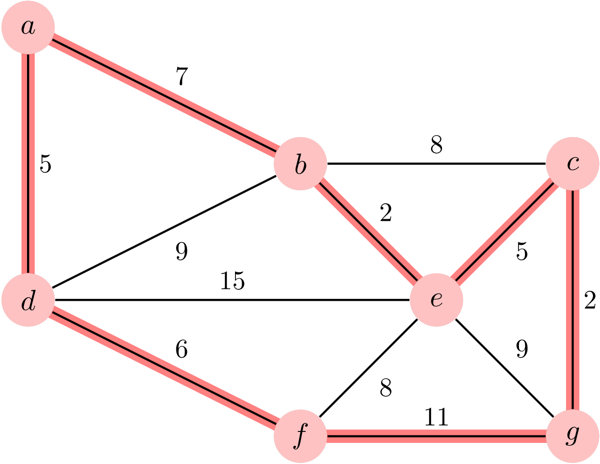


Exemple



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

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

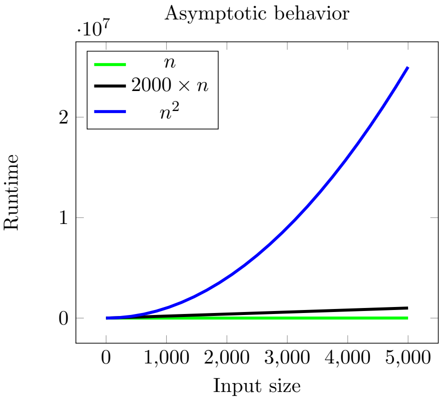
Example: The Sorting Integers problem

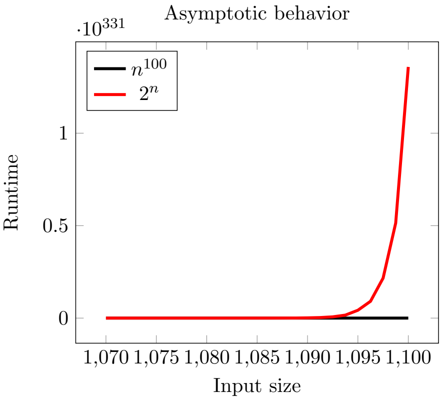
- Problem: sort a given sequence of n integers.
- Instance: a sequence of n integers
- A simple algorithm:
  - Scan the list to look for the smallest element
  - Swap it with the first position
  - Repeat for the list of remaining elements
- Example with the instance : 9, 3, 8, 7, 2
  - 2, 9, 3, 8, 7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

$x \lor \neg y \lor z$

$\neg x \lor \neg z$

$y \lor w$

$\neg w \lor \neg x$

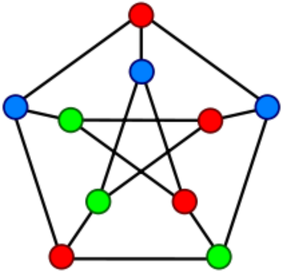
A possible solution:

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

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 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

- Propose a SAT model for this problem
  (hint $x \rightarrow y$ is equivalent to $\neg y \rightarrow x$ and both are translated into the clause $\neg x \vee y$).

The Example of Graph Coloring: A Possible Model

Let x_i^k be the Boolean variable that is True iff node i is coloured with the colour k.

I Each node has to be colored with at least one color:

$$\forall i \in [1, n], x_i^1 \vee x_i^2 \ldots x_i^k$$

II If a node is coloured with a colour a, the other colours are forbidden:

$$\forall i \in [1, n], \forall a \neq b \in [1, k], : \neg x_i^a \vee \neg x_i^b$$

(This is a translation of $x_i^a \rightarrow \neg x_i^b$)

III Forbid two nodes that share an edge to be coloured with the same colour

$$\forall \{i, j\} \in E, \forall a \in [1, k] : \neg x_i^a \rightarrow \neg x_j^a$$

(This is a translation of $x_i^a \rightarrow \neg x_j^a$)

The Example of Graph Coloring: The Model Size

What is the (space) size of the model?

- $n \times k$ Boolean variables
- Constraints form I: n clauses with k literals each
- Constraints form II: $n \times k^2$ binary clauses
- Constraints form III: $m \times k$ binary clauses

The Example of Graph Coloring: The Minimization Version

- 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.

A Straightforward Approach

- Find a valid upper bound UB and a lower bound LB for k
- Run iteratively the decision version until converging to the optimal value
- Let's call $SAT(V, E, K)$ the SAT model of the decision version of the problem (i.e., can we find a valid colouring of $G(V, E)$ with k colours). Use $SAT(V, E, K)$ as an oracle within an iterative search. For instance:
  - **Decreasing linear Search:** Run iteratively $SAT(V, E, UB-1), SAT(V, E, UB-2), \ldots$ until the problem is unsatisfiable. The last satisfiable value of k is the optimal value
  - **Binary search:** Run iteratively $SAT(V, E, z)$ as long as $UB > LB$ where $z = \lceil (UB - LB)/2 \rceil$. If the result is satisfiable, then and $UB \leftarrow z$ otherwise $LB \leftarrow z$

Upper/Lower Bound?

- Upper bound: For instance, we can run the following iterative greedy algorithm:
  - Each vertex v is considered non-coloured and has a portfolio S_v of available colours. The set is initially $\{1, 2, \ldots n\}$ for each vertex
  - At leach iteration, look for a non-coloured vertex v that has the greatest number of non coloured neighbours. Colour it with the smallest colour in S_v and remove its colour from all its neighbours.
  - The resulting colouring is valid and the the upper bound is the number of different colours used.
  - The run time complexity is $O(n^2 \times m)$
- Lower bound: Well, we can simply consider 2 as long as there is an edge. A more advanced one is to look for a clique in the graph.
- An alternative approach is to look for valid theoretical bounds in the literature.

Modelling Cardinality Constraints

- The general form of cardinality constraints is the following:

$$a \leq \sum_{1}^{n} x_i \leq b$$

  where a and b are positive integers and $x_1 \dots x_n$ are Boolean variables
- Cardinality constraints are everywhere!
- Many ways to encode such constraints. See for instance `https://www.carstensinz.de/papers/CP-2005.pdf`

Quadratic encoding for $\sum_1^n x_i = 1$

- At least one constraint:

$$x_1 \vee x_2 \ldots x_n$$

- at most one constraints:

$$\forall i, j : \neg x_i \vee \neg x_j$$

This generates one clause of size n and (n^2) binary clauses without introducing additional variables.

Linear encoding for $\sum_1^n x_i = 1$

New variables are added as follows: for $i \in [1, n], y_i$ is a new variable that is true iff $\sum_{l=1}^{l=i} x_l = 1$.

$$x_1 \vee x_2 \dots x_n$$

$$y_n^1$$

$$\forall i \in [1, n-1] : y_i \rightarrow y_{i+1}$$

$$\forall i \in [1, n-1] : y_i \rightarrow \neg x_{i+1}$$

$$\forall i \in [1, n] : x_i \rightarrow y_i$$

Size: n new variables, 1 $n-$ary clause and $3 \times n$ binary clauses,

Linear encoding for $\sum_1^n x_i \geq k$

New variables: $\forall z \in [0,k], \forall i \in [1,n], y_i^z \iff \sum_{l=1}^{l=i} x_l \geq z$

$$\forall i \in [0,n] : y_i^0 \leftarrow 1$$

$$y_1^1 \leftarrow x_1 \; and \; \forall z \in [2,k], y_1^z \leftarrow 0$$
$$y_n^k \leftarrow 1$$

$$\forall i \in [1,n], \forall z \in [1,k-1] : y_i^{z+1} \rightarrow y_i^z$$

$$\forall i \in [1,n-1], \forall z \in [1,k] : y_i^z \rightarrow y_{i+1}^z$$

$$\neg y_{i-1}^z \rightarrow \neg y_i^{z+1}$$

$$y_{i-1}^z \wedge x_i \rightarrow y_i^{z+1}$$

Linear encoding for $\sum_1^n x_i \geq k$

Size of the encoding:

- $\Theta(n \times k)$ variables
- $\Theta(n + k)$ unary clauses
- $\Theta(n \times k)$ binary clauses
- $\Theta(n \times k)$ ternary clauses

Linear encoding for $\sum_1^n x_i = k$?

- Encode $\sum_1^n x_i \geq k+1$
- Force y_n^{k+1} to be false and y_n^k to be true

Size of the encoding: Same as $\sum_1^n x_i \geq k$ (asymptotically)

Linear encoding for $\sum_1^n x_i \leq k$?

- Encode $\sum_1^n x_i \geq k + 1$
- Force y_n^{k+1} to be false

Size of the encoding: Same as $\sum_1^n x_i \geq k$ (asymptotically)

Linear encoding for $a \leq \sum_1^n x_i \leq b$?

- Encode $\sum_1^n x_i \leq b$
- Force y_n^a to be true

Size of the encoding: Same as $\sum_1^n x_i \geq b$ (asymptotically)

References I