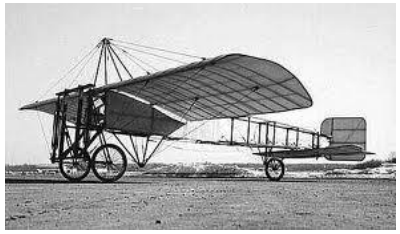
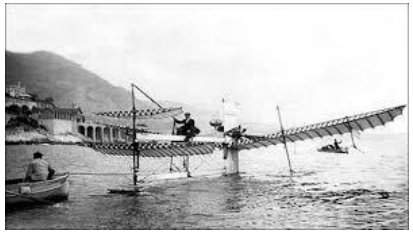
CentraleSupélec

Elements of Complex System Engineering

Antoine B. Rauzy
Department of Mechanical and Production Engineering (MTP)
Norwegian Science and Technology University (NTNU)
and
Chaire Blériot-Fabre
Centrale-Supélec, SAFRAN


CentraleSupélec

LECTURE 6. CONSTRAINTS

Notions:

- Constraint Satisfaction Problem
- Exploration Algorithms

CentraleSupélec

LECTURE 6. PART 1. INTRODUCTION

CentraleSupélec

Objective of this lecture

Many systems engineering questions can be expressed either as **constraint satisfaction problems** (CSP) or as **optimization under constraints problems**: scheduling/rostering, resource allocation, product configuration, circuit verification, DNA sequencing…

The objective of this lecture is:

- To introduce the notion of constraint satisfaction problem;
- To show, by means of examples, how to formulate a question about a system as a constraint satisfaction problem;
- To present the principles of **assessment algorithms**;
- To give a snapshot of the **limits** of these algorithms;
- To apply constraint solving techniques to a familiar example.

CentraleSupélec

Case Study: Nurse Rostering

This case study is taken from:

Peter Brucker, Rong Qu, Edmund Burke and Gerhard Post. *A Decomposition, Construction and Post-Processing Approach for Nurse Rostering.* In: Multidisciplinary International Conference on Scheduling : Theory and Applications, Aug, 2005, New York.

The problem at stake was to organize the schedule of the 16 nurses in an intensive care service of a Dutch hospital. This schedule had to be defined for a period of 5 weeks. The objective was to propose working hours for each of the nurses so that to match the needs of the service, the desiderata of the nurses, and the Dutch regulation. Moreover,

- 12 of the 16 nurses worked full time, i.e. 36 hours a week, 1 worked 32 hours a week and 3 worked 20 hours a week;
- The wishes of nurses should be exhausted except in case of strict impossibility.

Case Study: Nurse Rostering

Shift type	Start time	End Time	Demand						
			Mon	Tue	Wed	Thu	Fri	Sat	Sun
Early	07:00	16:00	3	3	3	3	3	2	2
Day	08:00	17:00	3	3	3	3	3	2	2
Late	14:00	23:00	3	3	3	3	3	2	2
Night	23:00	07:00	1	1	1	1	1	1	1

Day shifts spread over 9 hours but include 1 hour break. They are counted as 8 work hours.

- A nurse can start only one shift a day.
- A nurse cannot work more than 4 additional hours (compare to her regular service) over the 5 weeks period.
- A nurse cannot take more than 3 night shifts over the 5 weeks period.
- A nurse must have at least 2 free week-ends over the 5 weeks period.
- After 2 consecutive night shifts, a nurse must have at least 42 hours of rest.
- There must be at least 11 hours of rest over a period of 24 hours.
- A nurse must have at least 1 day off after 6 worked days.
- One of the full time nurse cannot work at night.

CentraleSupélec

Use Case: Nurse Rostering

In addition to the previous hard constraints, there is a number of constraints to be satisfied if possible. These constraints are weighted. E.g.

- During the week-ends, i.e. from Friday 23:00 to Monday 0:00, a nurse should either take two shifts, or none (weight 1000).
- Two or three night shifts must come consecutively and not be positioned at the end of a working period (weight 1000).
- After a series of day shifts, there should be at least two days off (weight 100).
- Nurses who work more than 30 hours a week should take 4 or 5 shifts a week (weight 10).
- Nurses who work less than 30 hours a week should take 2 or 3 shifts a week (weight 10).
- The number of consecutive “early” vacations should be 2 or 3 (weight 10).
- The number of consecutive “late” vacations should be 2 or 3 (weight 10).
- It is better not to take a “early” shift after “day” shift or a “late” shift and it is better not to take a “day” shift after a “late” shift (weight 5).
- It is better not to take a “night” shift after a “early” shift (weight 1).

CentraleSupélec

What to Do?

It is off course possible to try to organize the nurse rostering “by hand”, i.e. by means of a trials-and-errors approach, with a good deal of negotiation and diplomacy to make it work eventually. This is more than often what is done in practice, with some success. It is however also possible to express formally the problem and to ask the computer to give a hand. E.g.

We introduce the following variables.

$$X_{n,s,d,w} = \begin{cases} 1 \text{ if the nurse } n \text{ takes the shift } s \text{ the day } d \text{ of the week } w \\ 0 \text{ otherwise} \end{cases}$$

We write the various rules that must be obeyed as constraints over these variables. For instance, the rule “There must be 3 nurses working during the “early” shift the Monday of the first week” is translated into the following constraint.

$$\sum_{n=1}^{16} X_{n,early,monday,1} = 3$$

What to Do (bis)?

All the rules cannot be translated into simple equations. For instance, the rule "after two night shifts there must be at least 42 hours of rest", applied to nurse number 3 and to the Tuesday and Wednesday shifts of the second week is translated as follows.

$$X_{3,night,tuesday,2} = 1 \wedge X_{3,night,wenesday,2} = 1 \Rightarrow$$
$$X_{3,early,thursday,2} = 0 \wedge X_{3,day,thursday,2} = 0 \wedge X_{3,late,thursday,2} = 0$$
$$\wedge X_{3,early,friday,2} = 0 \wedge X_{3,day,friday,2} = 0 \wedge X_{3,late,friday,2} = 0$$

CentraleSupélec

What to Do (ter)?

Applying this method, we get a (big) set of constraints applying on a set of 16x4x7x5 = 2240 variables! The problem is now to find an assignment to 0 or 1 of these variables that satisfies all of the hard constraints and minimizes the sum of the weights of the soft constraints that are not satisfied.

Note that it is possible “just” to find an assignment that minimizes the sum of the weights of non satisfied constraints by giving an infinite weight to hard constraints.

Clearly, this cannot be done by hand.

Hence two questions:

- Is there a modeling formalism for this kind of problems?
- Is there an algorithm to solve this kind of problems?

The notion of **Constraint Satisfaction Problem** provides a yes answer to these two questions.

CentraleSupélec

LECTURE 6. PART 2. DEFINITIONS

CentraleSupélec

Constraint Satisfaction Problems

A **constraint satisfaction problem** is a pair (V, C) where:

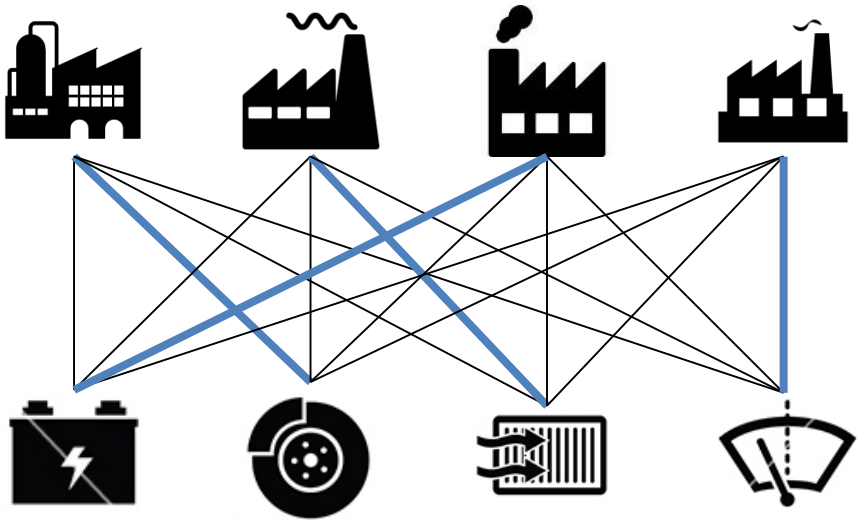
- V is a finite set of **variables**. Each variable $v \in V$ takes its value in (small) finite set of constants (Boolean, symbols, integers) called the **domain** of v and denoted by dom(v).
- C is a finite set of constraints on variables of V. Each constraint $c \in C$ applies to a subset $\{v_1, v_2, \ldots, v_k\}$ of V. It describes which k-tuples of values are admissible: $c \subseteq dom(v_1) \times dom(v_2) \times \ldots \times dom(v_k)$.

The k-tuples of a constraint can be given **explicitly** (by enumerating them) or **implicitly** (e.g. by means of an equation, as in the case study).

A **variable assignment** in a function from variables of V to their domains. An assignment is a **solution** of (V, C) if it satisfies all of the constraints of C.

Example: a Matching Problem*

Assign four suppliers S1, S2, S3, S4 to four parts such that each supplier provides on one part and each part is produced by one supplier.
Effectiveness of production is given by the following table (e.g., supplier S1 produces part P1 with effectiveness 7).
The total effectiveness must be 19 at least.



	P1	P2	P3	P4
S1	7	1	3	4
S2	8	2	5	1
S3	4	3	7	2
S4	3	1	6	3

(*) This example is borrowed from Hana Rudová "Constraint Programming and Scheduling -- Materials from the course taught at HTWG Constanz, Germany"

Example: a Matching Problem

Variables:

$S1 \in \{1, 2, 3, 4\}$ $E1 \in \{1, 3, 4, 7\}$
$S2 \in \{1, 2, 3, 4\}$ $E2 \in \{1, 2, 5, 8\}$
$S3 \in \{1, 2, 3, 4\}$ $E3 \in \{2, 3, 4, 7\}$
$S4 \in \{1, 2, 3, 4\}$ $E4 \in \{1, 3, 6\}$

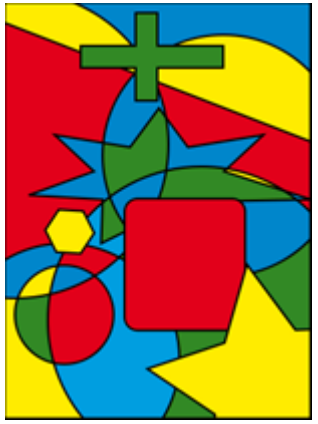
	P1	P2	P3	P4
S1	7	1	3	4
S2	8	2	5	1
S3	4	3	7	2
S4	3	1	6	3

Constraints (implicit):

$S1 \neq S2$, $S1 \neq S3$, $S1 \neq S4$
$S2 \neq S3$, $S2 \neq S4$,
$S3 \neq S4$,
$S1=1 \Rightarrow E1=7$, $S1=2 \Rightarrow E1=1$, $S1=3 \Rightarrow E1=3$, $S1=4 \Rightarrow E1=4$,
$S2=1 \Rightarrow E2=8$, $S2=2 \Rightarrow E2=2$, $S2=3 \Rightarrow E2=5$, $S2=4 \Rightarrow E2=1$,
$S3=1 \Rightarrow E3=4$, $S3=2 \Rightarrow E3=3$, $S3=3 \Rightarrow E3=7$, $S3=4 \Rightarrow E3=3$,
$S4=1 \Rightarrow E4=3$, $S4=2 \Rightarrow E4=1$, $S4=3 \Rightarrow E4=6$, $S4=4 \Rightarrow E4=3$,
$E1+E2+E3+E4 \geq 19$

Digression: the 4 colors theorem*

Theorem: it is possible to color any map with four colors only such that two adjacent regions (regions with a border not reduced to a single point) are never colored alike.



This result was first conjectured by Francis Guthrie in 1852. Francis Guthrie was interested in coloring the map of England regions. Several candidate proofs of the results were published but later on shown flawed.

In 1976, Kenneth Appel and Wolfgang Haken claimed to get a correct proof. However, their proof raised a lot of concerns in the scientific community: for the first time, the proof required the help of a computer to study the 1478 critical cases (which required with the computers available at than time more than 1200 hours of intensive computation). The problem was then moved to into two validation problems:

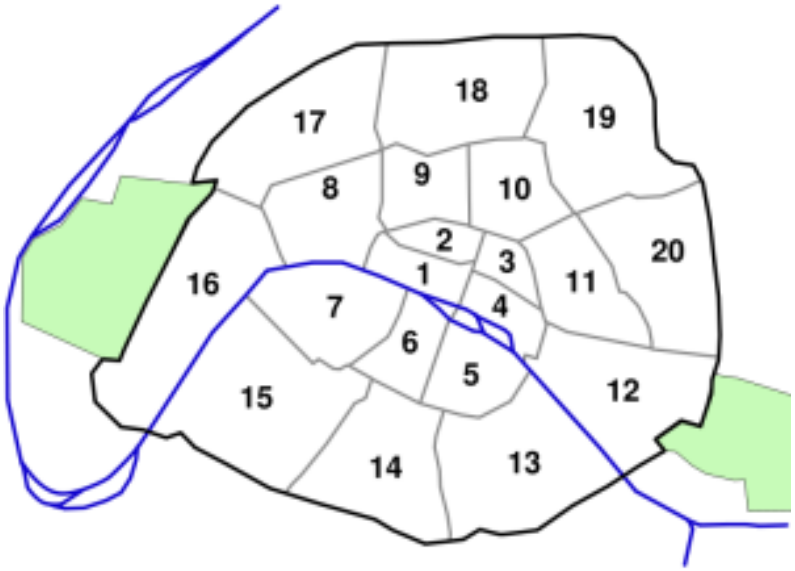
- To show that the exploration algorithm was correct;
- To show that its implementation into a computer code was correct.

U.S. postal stamp

(*) source : Wikipédia

Colors of Paris

Problem: Color the map of Paris districts with at most four colors such that no adjacent districts are colored alike.
Method: Formulate the problem as a constraint satisfaction problem.



- A variable per district D1,…, D20.
- Same domain for all variables, e.g. {green, red, yellow, blue}.
- Inequality constraints for adjacent districts, e.g. D1≠D2, D1≠D3…

Question: Find an algorithm to solve this constraint satisfaction problem.

CentraleSupélec

LECTURE 6. PART 3. ALGORITHMS

Algorithm “Generate and Test”

There exists a quite simple algorithm to solve any constraint satisfaction problem: it consists in generating one by one all possible variable assignments and to test them until a solution is found.

GenerateAndTest(V, C):

    GenerateAndTest(∅, V, C)

GenerateAndTest(σ, V, C):

    if σ gives a value to all variables of V:

        if σ satisfies all constraints of C:

            exit with σ

    else:

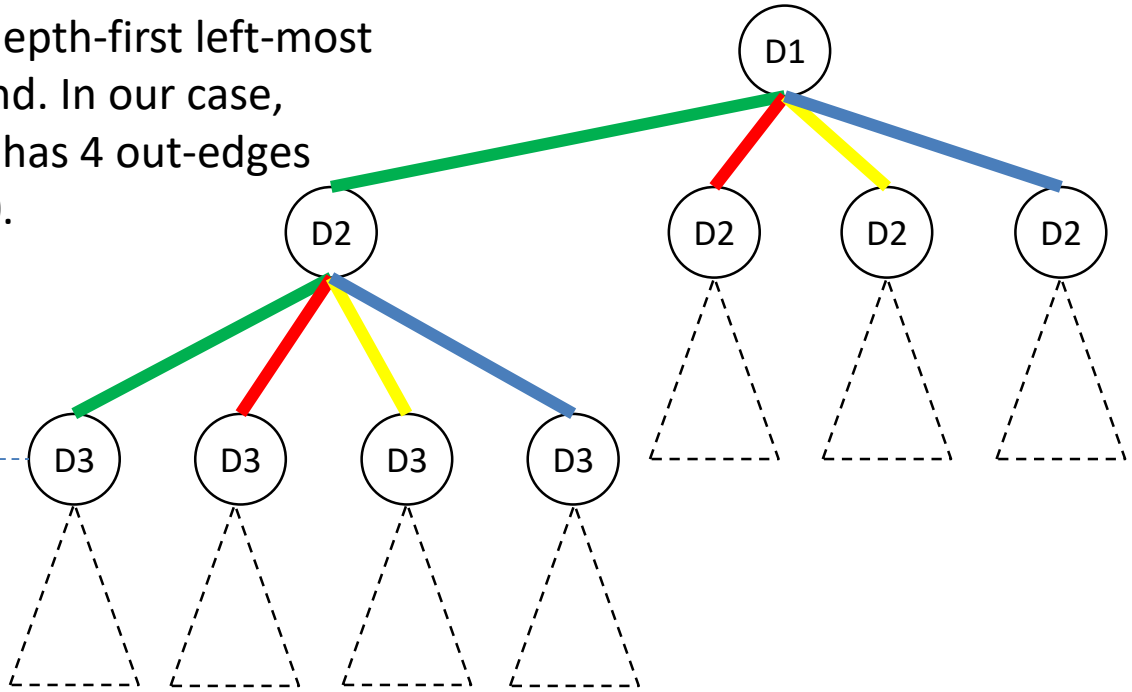
        select a variable v in V not assigned in σ

        forall constant c in dom(v):

            GenerateAndTest(σ ∪ [v ← c], V, C)

Colors of Paris

The algorithm “generate-and-test” consists in exploring a decision tree in a depth-first left-most way until a solution leaf is found. In our case, each internal node of the tree has 4 out-edges and the depth of the tree is 20.

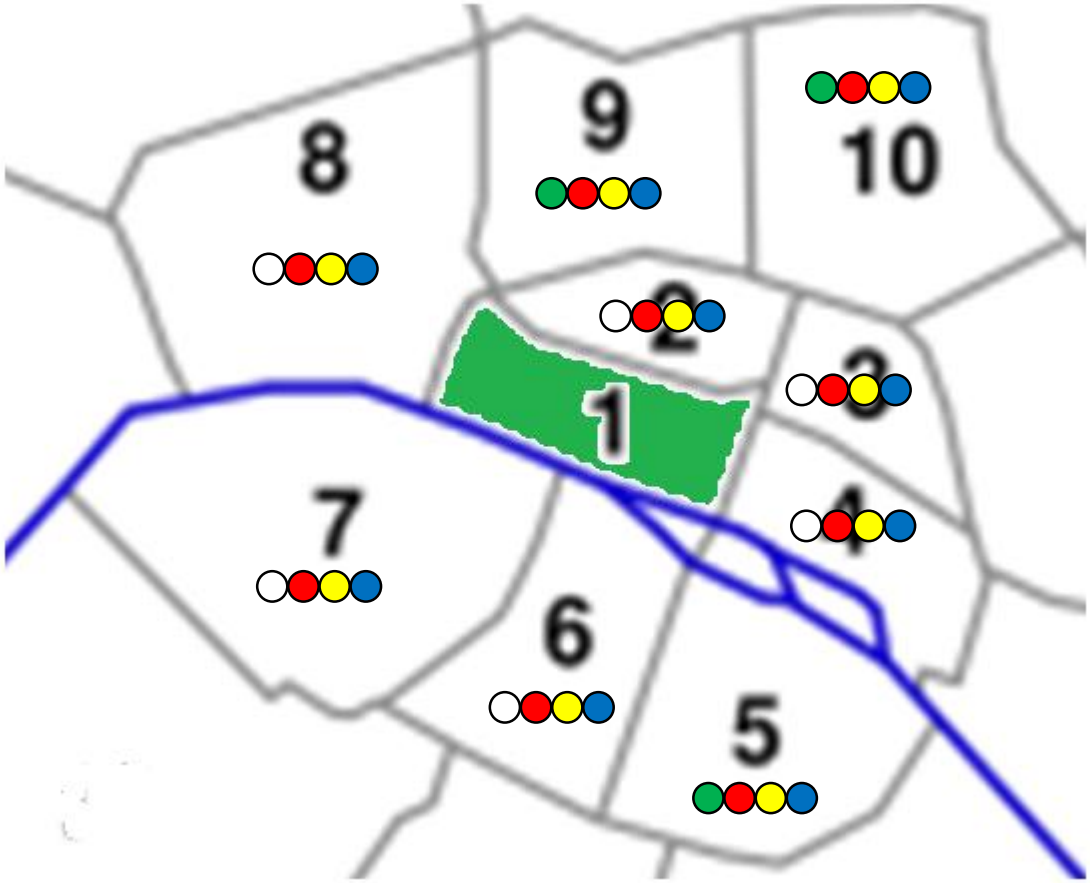


Remarks:

- If the four-color theorem is false, i.e. if there is no solution, the generate-and-test algorithm will explores 4^{20} branches!
- The efficiency of the algorithm could be (significantly) improved by stopping the exploration as soon as one of the constraints is falsified, e.g. D1=green, D2=green falsifies D1≠D2.

Propagate

When value is assigned to a variable, it is possible to **propagate** immediately this assignment, i.e. to use constraints to **remove some candidate values** from still un-assigned variables.



If district 1 is colored in green, then districts 2, 3, 4 6, 7 and 8 cannot be colored in green. This value can be thus removed from their sets of candidate values.

Constraints: New Vision

Constraints should thus be seen as an **algorithmic mechanism** to **restrict domains** of variables.

Example: “Three nurses must work during the early shift of the Monday of the second week”

$$\sum_{n=1}^{16} X_{n,early,monday,2} = 3$$

If nurses number 1, 2 and 3 are working this shift (the corresponding variables are set to 1), then none of the other nurses will work this shift (the value 1 is removed from the domains of corresponding variables, which are therefore set to 0).

If none of the nurses number 1 to 13 is working during this shift (the corresponding variables are set to 0), then nurses 14, 15 and 16 must work this shift (the value 0 is removed from the domains of the corresponding variables, which are therefore set to 1).

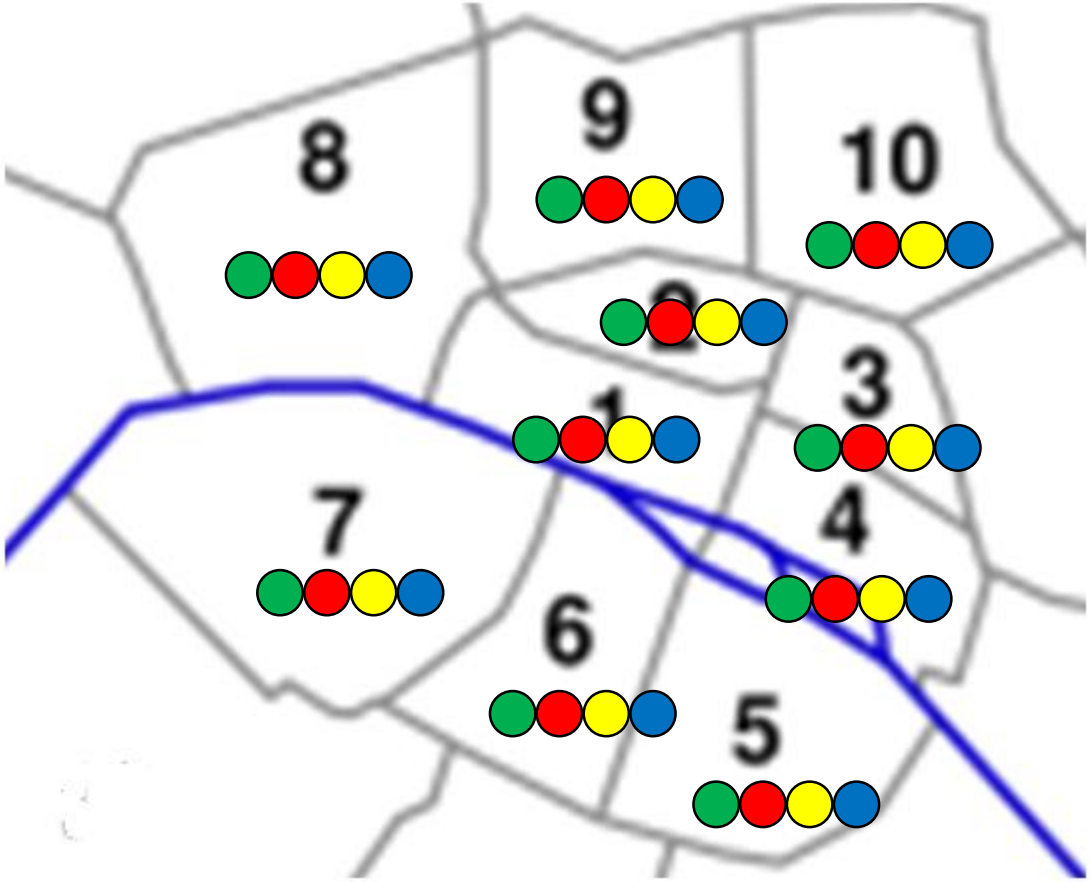
Exploration Algorithm

```
Explore(σ, V, C):
    if σ falsifies one of the constraints of C:
        return None
    else if σ satisfies all constraints of C:
        return σ
    else:
        select a variable v in V not assigned in σ and a value c in dom(v)
        σ' ← Propagate(σ|[dom(v) ← {c}], V, C)
        result ← Explore(σ', V, C)
        if result=None:
            σ'' ← Propagate(σ|[dom(v) ← dom(v)\{c}], V, C)
            result ← Explore(σ'', V, C)
        return result
```

Assigns the value c to v and propagate

Removes the value c from dom(v) and propagate

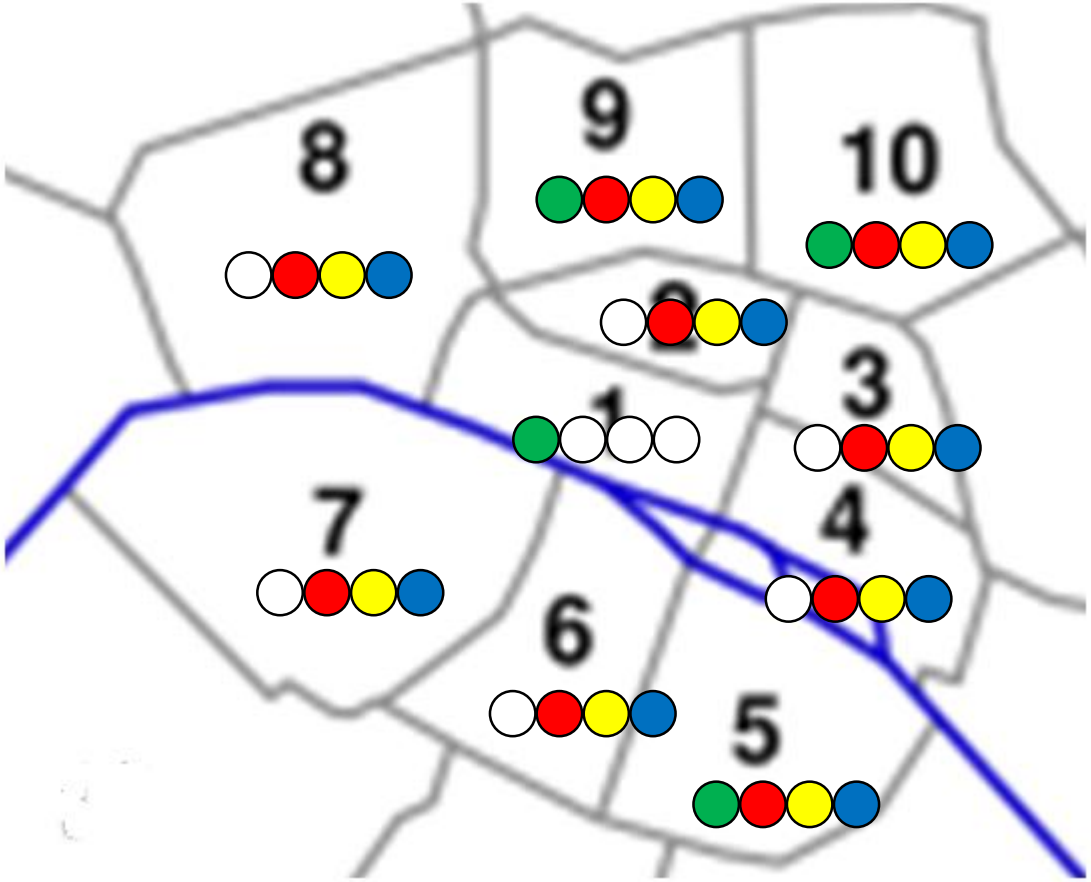
Colors of Paris



select (D1, green)

dom(D1) = {green}
dom(D2) = {red, yellow, blue}
dom(D3) = {red, yellow, blue}
dom(D4) = {red, yellow, blue}
dom(D6) = {red, yellow, blue}
dom(D7) = {red, yellow, blue}
dom(D8) = {red, yellow, blue}

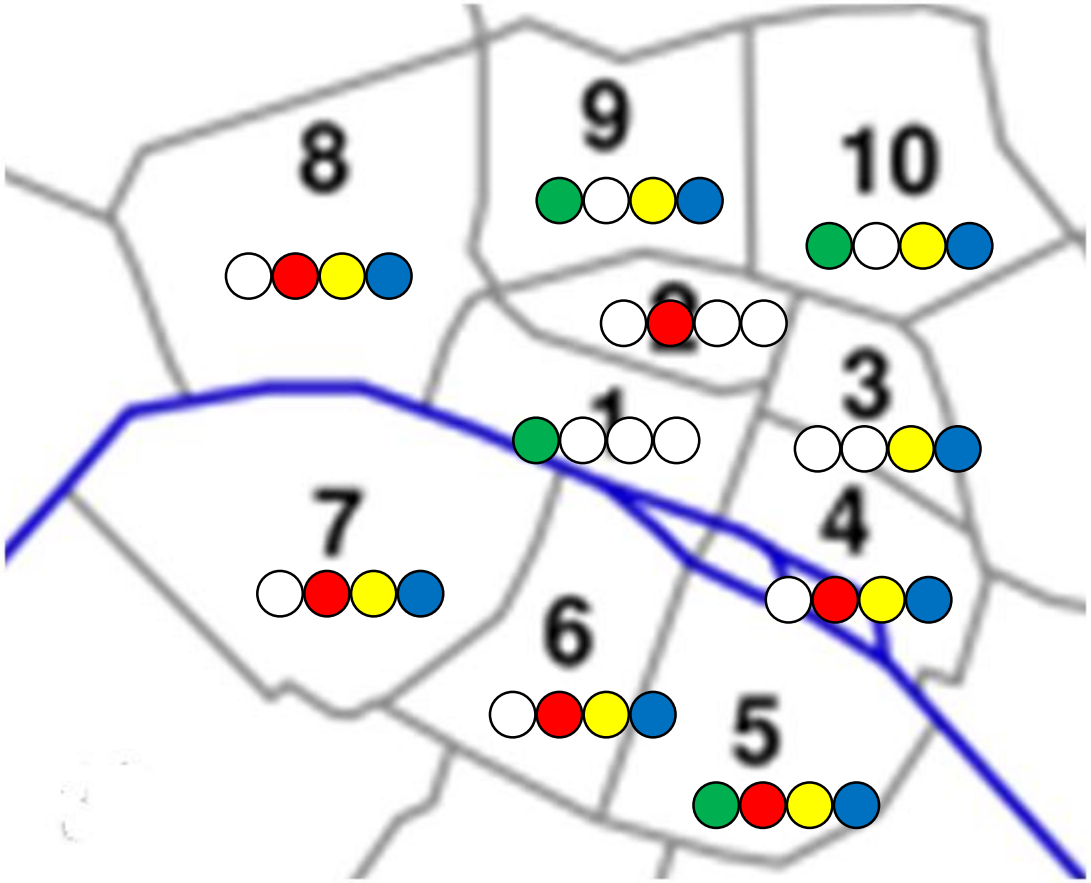
Colors of Paris



select (D2, red)

dom(D2) = {red}
dom(D3) = {yellow, blue}
dom(D8) = {red, yellow, blue}
dom(D9) = {green, yellow, blue}
dom(D10) = {green, yellow, blue}

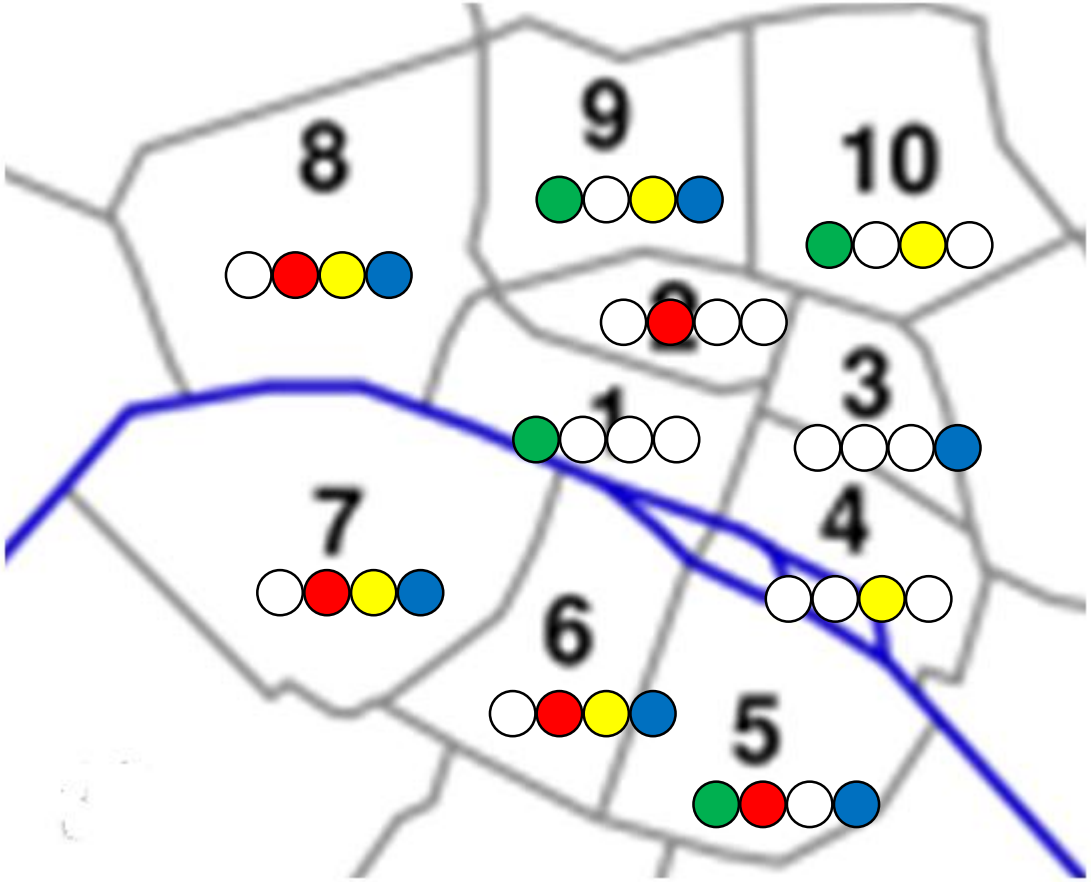
Colors of Paris



select (D4, yellow)

dom(D4) = {yellow}
dom(D3) = {blue}
dom(D5) = {green, red, blue}
dom(D10) = {green, yellow}

Colors of Paris



and so on…

CentraleSupélec

Selection Heuristics

The **size** of the **exploration tree** depends on how efficient the propagation mechanism is. It depends also of the **choice** of the **next pair (variable, value)** to be considered.

There is **no method** (but to try everything) that warranties that the explored tree is the smallest possible. There is even no method that warranties to get a small enough tree (whatever it means).

Therefore, heuristics are used. A **heuristic** is a simple and cheap method to select the next pair (variable, value) that gives (hopefully) good results in practice.

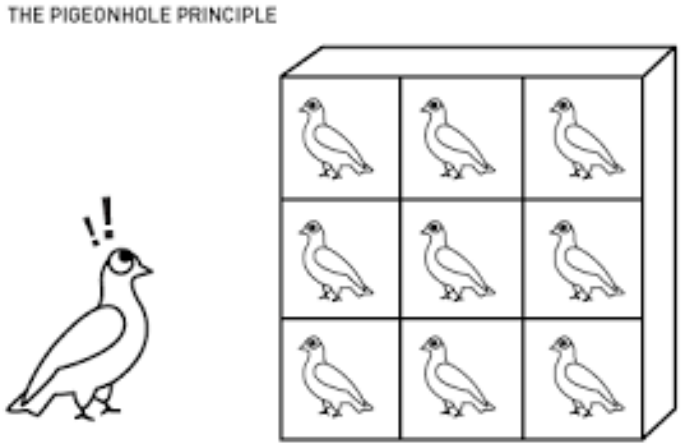
For instance:

- Select the variable with the smallest domain.
- Select a variable and a value showing up in the constraint with the least number of satisfying assignments.

The Pigeon-Hole Principle

One of main problems exploration algorithms are facing can be illustrated on our use case: assume that, at some step of the exploration, the algorithm has to fill 4 shifts with 3 nurses. As humans, we see immediately that's impossible. But the computer does not "see" anything. So it assigns the first nurse to the first shift, finds no solution, backtracks, assigns the first nurse to second shift, finds no solution, backtracks and so on.

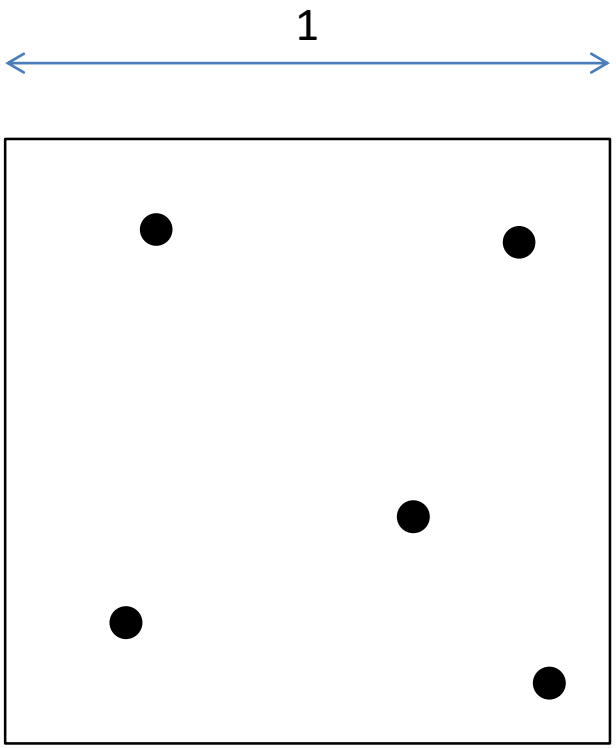
This problem is an application of a general principle strangely called "**Pigeon-Hole principle**". It can be stated as follows.
It is not possible to fill n holes with n+m pigeons (n, m>0) such that no hole is occupied by more than one pigeon.



A priori, this looks quite easy to detect such situations. In practice, it is not: **breaking symmetries** of a problem is a **difficult task** because the **cost of the detection**, including in the cases where there is no symmetry, should not exceed the cost of the brute force exploration algorithm.

Digression: a small theorem in geometry

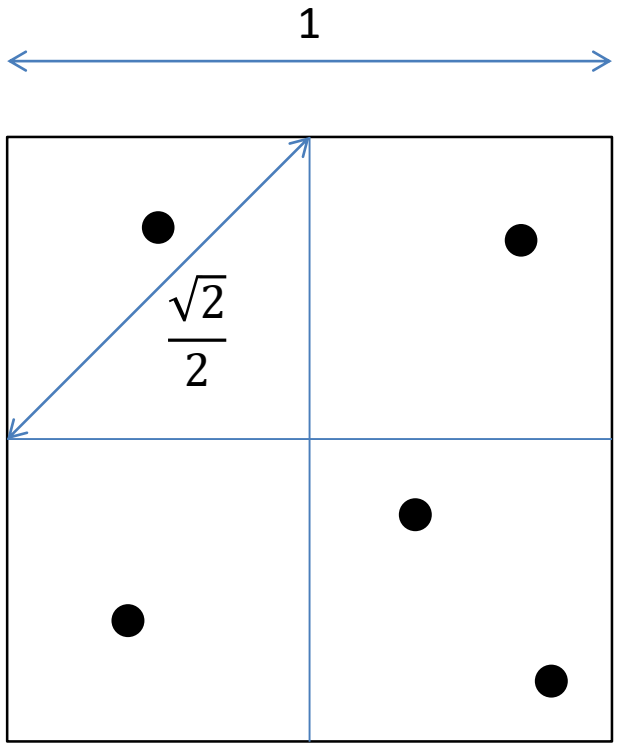
Theorem: Consider the unity square and 5 points located anywhere in this square. Then, at least of two these points are distant one another of less than $\sqrt{2}/2$.



$$\frac{\sqrt{2}}{2} \cong 0.707106781$$

Digression: a small theorem in geometry

Theorem: Consider the unity square and 5 points located anywhere in this square. Then, at least of two these points are distant one another of less than $\sqrt{2}/2$.



Proof: just split the square into 4 sub-squares with sides of length ½ as shown on the figure. Then, according to the pigeon-hole principle, at least two of the points will end up into the same sub-square.
But the maximal distance of two points in this square is the diagonal whose size is:

$$\sqrt{{}^{1}/_{2}{}^{2} + {}^{1}/_{2}{}^{2}} = \sqrt{2}/2$$

Q.E.D.

CentraleSupélec

Back to Constraints

A practical solution consists in developing specialized constraints that detect and remove as early as possible impossible assignments. E.g.

$\sum_{i=1}^{n} a_i X_i = k$ (with a_i's integral coefficients), it works also with <, >, ≤, ≥.

alldifferent(X_1, X_2,…, X_n): variables X_1, X_2,…, X_n must all take different values.
As soon as one take a value v, this value is removed from the domains of the other variables.

element(N, [X1, X2,…, Xn], V): the N-th element of the list equals V.

and so even more complex ones…

CentraleSupélec

Beyond Exploration Algorithms

- It is possible to adjust the exploration algorithm to find an **optimal solution**, i.e. the solution optimizing some criterion (such an algorithm is for instance required to solve the "Nurse Rostering" use case). The corresponding algorithms are called "**branch-and-bound**".

- Both exploration and branch-and-bound algorithms are **exhaustive**: if a solution exists, they will find it, possible after a long time.

- Sometimes, exhaustive search is too costly. **Incomplete algorithms** are then used. There are numerous principles to design such algorithms. These principle are called **meta-heuristics**. Example of popular meta-heuristics: **taboo search**, **simulated annealing**, **genetic algorithms**, **ant colony algorithms**…

CentraleSupélec

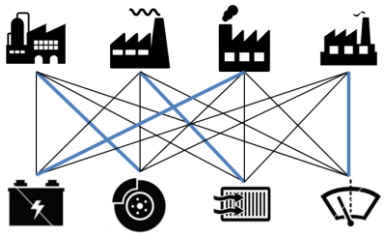
LECTURE 6. PART 4.
CONSTRAINTS AS A MODELING LANGUAGE

Modeling Language

There exist several formalisms and tools to state and solve constraint satisfaction problems (e.g. ILOG Solver). Constraint satisfaction problems have basically two parts:

- Variables declarations

$S1 \in \{1, 2, 3, 4\}$ $E1 \in \{1, 3, 4, 7\}$
$S2 \in \{1, 2, 3, 4\}$ $E2 \in \{1, 2, 5, 8\}$
$S3 \in \{1, 2, 3, 4\}$ $E3 \in \{2, 3, 4, 7\}$
$S4 \in \{1, 2, 3, 4\}$ $E4 \in \{1, 3, 6\}$



	P1	P2	P3	P4
S1	7	1	3	4
S2	8	2	5	1
S3	4	3	7	2
S4	3	1	6	3

- Constraints:

alldifferent(S1, S2, S3, S4)
$S1=1 \Rightarrow E1=7$, $S1=2 \Rightarrow E1=1$, $S1=3 \Rightarrow E1=3$, $S1=4 \Rightarrow E1=4$,
$S2=1 \Rightarrow E2=8$, $S2=2 \Rightarrow E2=2$, $S2=3 \Rightarrow E2=5$, $S2=4 \Rightarrow E2=1$,
$S3=1 \Rightarrow E3=4$, $S3=2 \Rightarrow E3=3$, $S3=3 \Rightarrow E3=7$, $S3=4 \Rightarrow E3=3$,
$S4=1 \Rightarrow E4=3$, $S4=2 \Rightarrow E4=1$, $S4=3 \Rightarrow E4=6$, $S4=4 \Rightarrow E4=3$,
$E1+E2+E3+E4 \geq 19$

It may be convenient however to introduce domain declarations to avoid repeating the same domain many times (e.g. {1, 2, 3, 4})

Modeling Language

```
problem Matching
   domain SupplierDomain [1, 4]
   SupplierDomain S1
   SupplierDomain S2
   SupplierDomain S3
   SupplierDomain S4
   {1, 3, 4, 7} E1
   {1, 2, 5, 8} E2
   {2, 3, 4, 7} E3
   {1, 3, 6} E4
   constraints
      …
end
```

Two types of domains:

- Ranges, e.g. [1, 4]
- Sets, e.g. {1, 3, 6}

Domains can be declared (given a name) and reused

Variables are declared by preceding their name with a domain, or the name of a domain

Modeling Language: Domains and Variables

```
problem Matching
   domain SupplierDomain [1, 4]
   SupplierDomain S1
   SupplierDomain S2
   SupplierDomain S3
   SupplierDomain S4
   {1, 3, 4, 7} E1
   {1, 2, 5, 8} E2
   {2, 3, 4, 7} E3
   {1, 3, 6} E4
   constraints
      …
end
```

Two types of domains:

- Ranges, e.g. [1, 4]
- Sets, e.g. {1, 3, 6}

Domains can be declared (given a name) and reused

Variables are declared by preceding their name with a domain, or the name of a domain

Modeling Language: Constraints

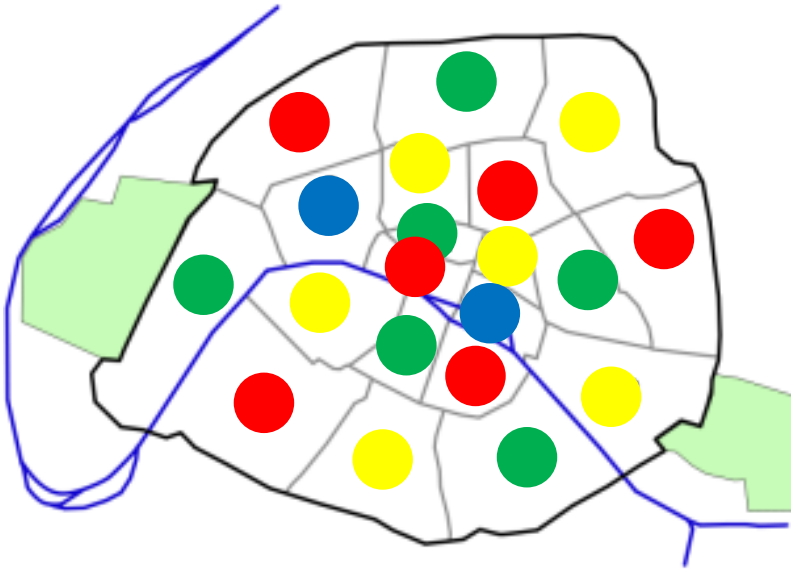
3 types of constraints:

- Alldifferent constraints: the given variables must take different values
- Clauses:
  - At least one of literals must be verified
  - Literals in the form V ® c , where ® is =, !=, <, >, <= or >=
- Linear inequalities:
  - a1*V1 + a2*V2 + … + ak*Vk ® c, where ® is =, !=, <, >, <= or >=
  - "+" are mandatory. The coefficient can be omitted (when equal to 1)

```
problem Matching
   …
   constraints
      alldifferent S1, S2, S3, S4;
      or S1!=1, E1=7;
      or S1!=2, E1=1;
      …
      or S4!=4, E4=3;
      E1+E2+E3+E4 >= 19;
end
```

Colors of Paris

Problem: Color the map of Paris districts with at most four colors such that no adjacent districts are colored alike.
Method: Formulate the problem as a constraint satisfaction problem.

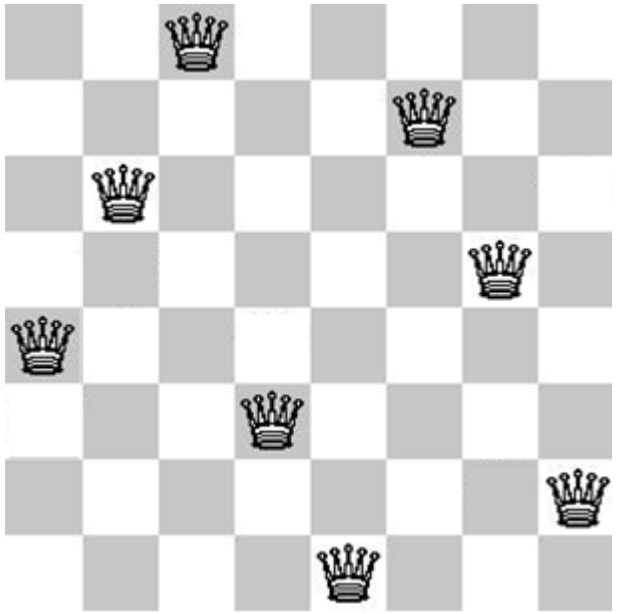


- A variable per district D1,…, D20.
- Same domain for all variables, e.g. {green, red, yellow, blue}.
- Inequality constraints for adjacent districts, e.g. D1≠D2, D1≠D3…

Question: Find an algorithm to solve this constraint satisfaction problem.

8 Queens

```
problem queens8
    domain Queen [1, 8]
    Queen Q1
    …
    Queen Q8
    constraints
        alldifferent Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8;
        Q1 + -1*Q2 !=1;
        Q1 + -1*Q2 !=-1;
        Q1 + -1*Q3 !=2;
        Q1 + -1*Q3 !=-2;
        ..
        Q7 + -1*Q8 != 1;
        Q7 + -1*Q8 != -1;
end
```



CentraleSupélec

sherlock.py

The Python program "sherlock.py" reads a constraint satisfaction problem in a text file and seeks for a solution.

As most, if not all, programs designed for this course, it is structured as follows.

1. Definition of data structures. In this lecture constraint satisfaction problem.
2. Classes that makes it possible to read (Reader) and write (Writer) data structures into text files as well as results of calculations.
3. Classes that implement the calculation(s) of interest (Calculator).
4. Main part of the program where the classes are instanced and the calculations performed.

You can run the program either in a command shell or using IDLE.

CentraleSupélec

LECTURE 6. PART 5.
WRAP-UP & ASSIGNMENT

CentraleSupélec

Wrap-Up

- Lots of **decision making problems** can be formulated as **constraint satisfaction problems**.
- **Solving** constraint satisfaction problems is done by means of **algorithms** that embed three main mechanisms:
  - A general **exploration mechanism**;
  - A **propagation mechanism** relying on **specialized constraints**;
  - A variable/value **selection heuristic**.
- These algorithms, as elaborated they can be, are facing the combinatorial explosion of the state space. A model results thus always of a **tradeoff** between the **accuracy** of the model and the **ability to solve** it.
- To face this phenomenon, **non-exhaustive algorithms** are sometime used. These algorithms are called **meta-heuristics**.

Assignment: Problem 1

Use “sherlock.py” to solve the following Sudoku grid.

				2				1
						5	8	
			8	5		7	4	
		2	5		6			4
3			4		2			7
6			9		7	1		
	2	9		7	1			
	7	6						
8				4				

CentraleSupélec

Assignment: Problem 2

You need to organize the shifts of 7 employees (Adna, Bekka, Carsten, Dina, Egil, Finn, Gill) to monitor operations of an offshore platform over one week (7 days). There are 3 types of shifts: morning, afternoon and night.

3 employees are required for morning shifts, 2 to 3 for afternoon shifts and 1 to 2 for night shifts.

Each employee must do 2 to 3 morning shifts, 1 to 3 afternoon shifts and 1 to 2 night shifts, but no less than 4 shifts and no more than 6 shifts.

It is not possible for an employee to take two consecutive shifts.

Morning shifts are payed as 10 work hours, afternoon shifts 11 hours and night shifts 13 hours.

Use “sherlock.py” to solve to find an acceptable rostering and to determine its costs (total number of payed hours).

How would you minimize this cost (still using “sherlock.py”)?

Recommend Readings

Reference books on constraints:

There exists a vast scientific and technical literature on constraint. Two historical refrences:

Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press Inc (août 1993). ISBN-10: 0127016104. ISBN-13: 978-0127016108

Rina Dechter. *Constraint Processing*. The Morgan Kaufmann Series in Artificial Intelligence. May 19, 2003. ISBN-10: 1558608907. ISBN-13: 978-1558608900.

Francesca Rossi, Peter Van Beek, Toby Walsh (editors): Handbook of Constraint Programming, Elsevier, 2006

CentraleSupélec

Louis Charles Joseph Blériot (1872 -1936) is an airplane designer and one of the pioneer pilot of French aviation. He has been the first to cross the channel on July the 25th onboard of the Blériot XI.
He graduated from Ecole Centrale de Paris

Henri Marie Léonce Fabre (1882 -1984) is a French engineer and pilot. He invented the seaplane in 1910.
He graduated from Supélec.