Constraint Programming

Lecture 1. Constraint Satisfaction Problems

Ruslan Sadykov

INRIA Bordeaux — Sud-Ouest

6th January 2022

Course organisation

- 5 lectures/exercice labs + 5 computer labs.
- Exercice labs : solving exercises on a paper
- 2-3 labs : introduction to a CP solver
- 2-3 labs : **project**
- Evaluation : 50% of the mark for the project + 50% of the mark for the exam (TD notés).
- Course web-page :

  www.math.u-bordeaux.fr/~sadykov/teaching/MSE3315C/

Course organisation

- 5 lectures/exercice labs + 5 computer labs.
- Exercice labs : solving exercises on a paper
- 2-3 labs : introduction to a CP solver
- 2-3 labs : **project**
- Evaluation : 50% of the mark for the project + 50% of the mark for the exam (TD notés).
- Course web-page :

  `www.math.u-bordeaux.fr/~sadykov/teaching/MSE3315C/`

Course organisation

- 5 lectures/exercice labs + 5 computer labs.
- Exercice labs : solving exercises on a paper
- 2-3 labs : introduction to a CP solver
- 2-3 labs : **project**
- Evaluation : 50% of the mark for the project + 50% of the mark for the exam (TD notés).
- Course web-page :

  www.math.u-bordeaux.fr/~sadykov/teaching/MSE3315C/

Course organisation

- 5 lectures/exercice labs + 5 computer labs.
- Exercice labs : solving exercises on a paper
- 2-3 labs : introduction to a CP solver
- 2-3 labs : **project**
- Evaluation : 50% of the mark for the project + 50% of the mark for the exam (TD notés).
- Course web-page :

  `www.math.u-bordeaux.fr/~sadykov/teaching/MSE3315C/`

Course organisation

- 5 lectures/exercice labs + 5 computer labs.
- Exercice labs : solving exercises on a paper
- 2-3 labs : introduction to a CP solver
- 2-3 labs : **project**
- Evaluation : 50% of the mark for the project + 50% of the mark for the exam (TD notés).
- Course web-page :

  `www.math.u-bordeaux.fr/~sadykov/teaching/MSE3315C/`

Course organisation

- 5 lectures/exercice labs + 5 computer labs.
- Exercice labs : solving exercises on a paper
- 2-3 labs : introduction to a CP solver
- 2-3 labs : **project**
- Evaluation : 50% of the mark for the project + 50% of the mark for the exam (TD notés).
- Course web-page :

  `www.math.u-bordeaux.fr/~sadykov/teaching/MSE3315C/`

Contents

Introduction

Applications of Constraint Programming

Constraint Satisfaction Problems

Modelling examples

Solving Constraint Satisfaction Problems

Solving technology

A solving technology offeres methods and tools for :

- **Modelling** constraint problems in declarative
  and/or
- **Solving** constraint problems intelligently

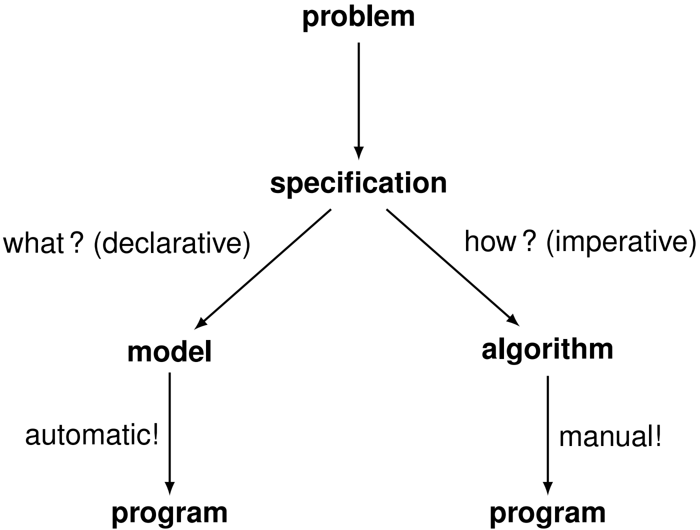
Search :	Explore the space of candidate solutions
Inference :	Reduce the space of candidate solutions
Search :	Exploit solutions to easier (sub)problems

A solver is a software that takes a model as input and tries to solve the modelled problem.

Combinatorial (=discrete) optimisation covers satisfaction and optimisation problems, for variables over *discrete sets*

Source : Pierre Flener

Modelling vs. Programming



Source : Pierre Flener

Examples of solving technologies

General-purpose solvers, taking a model as input :

- Boolean satisfiability (SAT)
- SAT modulo theories (SMT)
- (Mixed) integer linear programming (IP and MIP)
- Constraint Programming (CP)
- ...
- Hybrid technologies

Techniques, usually without modelling and solvers :

- Dynamic programming (DP)
- Greedy algorithms
- Approximation algorithms
- Generic algorithms (GA)
- ...

Source : Pierre Flener

Constraint Programming Technology

Constraint Programming (CP) offeres methods and tools for :

- **Modelling** constraint problems in a high-level language and
- **Solving** constraint problems intelligently by :
  - either default search upon pushing a button
  - or systematic search guided by user-given strategies
  - or local search guided by user-given (meta-)heuristics
  - or hybrid search

  plus inference, called propagation, but little relaxation.

Slogan of CP :

Constraint Program = Model [+ Search]

Source : Pierre Flener

Limitations of CP

CP is **definitely not**

- a ~~**magic method**~~
  - A priori, it is not better than other methods (integer linear programming, dynamic programming, local search, etc...)
  - It depends on the problem type !
- a ~~« **press button** »~~ method, at least for the moment
  - It is necessary to understand the method (what is going on « inside » it)
  - It is necessary to « guide » the solution

Source : Antoine Jeanjean

Objectives of the course

- To know for which (classes of) problems the CP methods is good
- To know how to model efficiently these problems
- To know which modelling languages and CP solvers exist and to know how to use them
- To understand how these solvers work inside

Particularities of CP

- We work with decision problems — constraint satisfaction problems (CSP)
  (if an optimisation problem, a series of CSPs is solved)
- Large modelling possibilities
  (non-linear, logical, explicit constraints)
- Use of problem constraints in an active way to limit the search space
  (Additional constrains may make a problem easier)

Particularities of CP

- We work with decision problems — constraint satisfaction problems (CSP)
  (if an optimisation problem, a series of CSPs is solved)
- Large modelling possibilities
  (non-linear, logical, explicit constraints)
- Use of problem constraints in an active way to limit the search space
  (Additional constrains may make a problem easier)

Particularities of CP

- We work with decision problems — constraint satisfaction problems (CSP)
  (if an optimisation problem, a series of CSPs is solved)
- Large modelling possibilities
  (non-linear, logical, explicit constraints)
- Use of problem constraints in an active way to limit the search space
  (Additional constrains may make a problem easier)

Introductive example — Sudoku

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

- There are 81 cells where a digit from 1 to 9 can be put
- We need to put digits to cells in such a way that every row (column, or a block of 9 cells) contains different numbers

Introductive example — Sudoku

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

- There are 81 cells where a digit from 1 to 9 can be put
- We need to put digits to cells in such a way that every row (column, or a block of 9 cells) contains different numbers

Introductive example — Sudoku

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

- There are 81 cells where a digit from 1 to 9 can be put
- We need to put digits to cells in such a way that every row (column, or a block of 9 cells) contains different numbers

Introductive example — Sudoku

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

- There are 81 cells where a digit from 1 to 9 can be put
- We need to put digits to cells in such a way that every row (column, or a block of 9 cells) contains different numbers

We have just (almost) formulated a Constraint Satisfaction problem(!)

Introductive example — Sudoku

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

- There are 81 cells where a digit from 1 to 9 can be put
- We need to put digits to cells in such a way that every row (column, or a block of 9 cells) contains different numbers

We have just (almost) formulated a Constraint Satisfaction problem(!)

In CP, the problem is solved more or less the same way you solve a sudoku(!)

Contents

Introduction

Applications of Constraint Programming

Constraint Satisfaction Problems

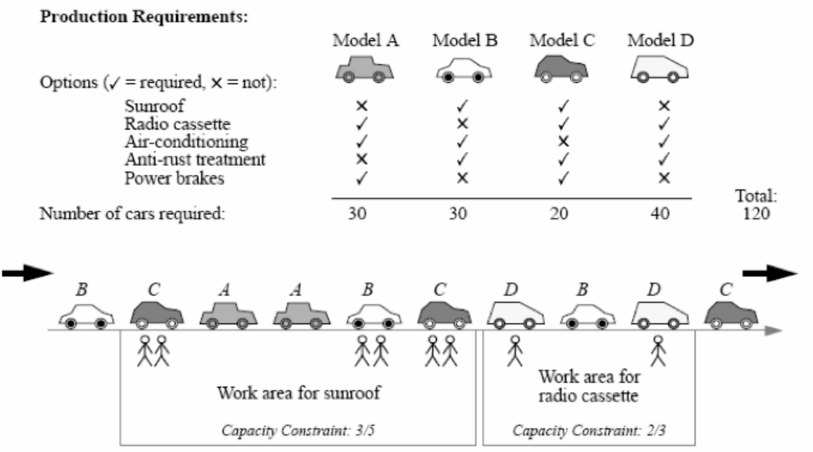
Modelling examples

Solving Constraint Satisfaction Problems

Application domains

- Location problems
- Diagnosis and verification
- Planning problems
- Scheduling and timetabling problems
- Cutting and packing
- Logistic problems

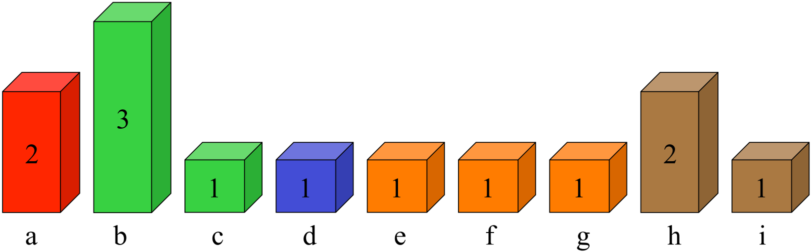
Real-life application I — Car Sequencing



Source : Alan M. Frisch

Real-life application II — Steel mill slab design

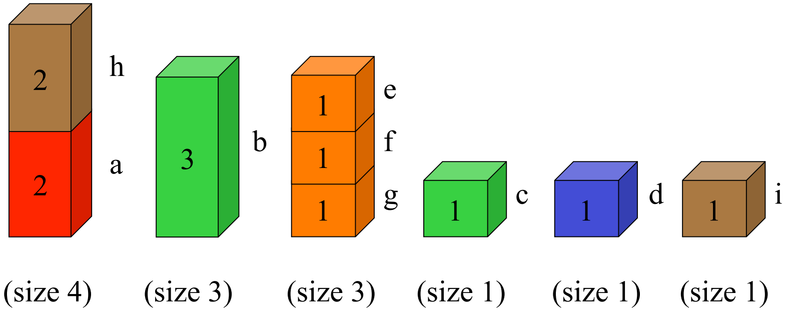
- The mill can make σ different slab sizes.
- For each order $j \in J$, we know a *colour* (route through the mill) and a *weight*
- We need to pack orders onto slabs such that the total slab capacity is minimized subject to
  - capacity (slab size) constraints
  - colour constraints (no more than p colours per slab)



Source : Alain Frisch

Steel mill slab design — an example solution

- Slab sizes : $\sigma = \{1, 2, 4\}$.
- 9 orders
- 5 different colours
- Maximum number of different colours per slab is 2



Source : Alain Frisch

Real-life application III — Sports scheduling

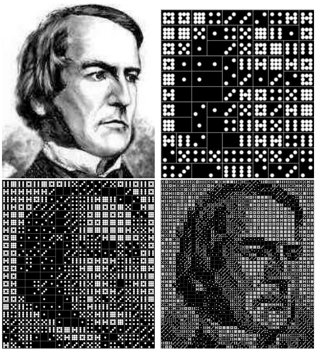
There several sport teams. In the championship, each team should play with each other team. We need a schedule : for each round we need to determine the pairs of teams playing with each other. We can have additional constraints.

Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7
1 vs 8	2 vs 8	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
2 vs 3	1 vs 7	3 vs 8	5 vs 7	1 vs 4	6 vs 8	5 vs 6
4 vs 5	3 vs 5	1 vs 6	4 vs 8	2 vs 6	2 vs 7	7 vs 8
6 vs 7	4 vs 6	2 vs 5	1 vs 2	5 vs 8	3 vs 4	1 vs 3

A « fun » application — Domino portraits

Aim : find a good approximation of an image using the dominos from an integer number of boxes.

Example on the right : A portrait of *George Boole*, and then a sequence of domino portraits generated using 1, 4, 16 domino boxes.



Source : (Cambazard, Horan, O'Mahony, O'Sullivan, 2008)

Success stories by CP users and contributors

AIMMS — AMPL — BOUYGUES

cadence — CISCO — FICO

Google — IBM — intel

JEPPESEN A BOEING COMPANY — NASA — ORACLE

QUINTIQ — RedPrairie — RENAULT

SAP — SIEMENS — TACTON

THALES — XEROX — . . .

Success stories : CP = **technology of choice** in scheduling, configuration, personnel rostering, timetabling, **...**

Source : Pierre Flener

Real-life application at Bouygues e-lab

- Table plans for the group conferences
- Planning for interior works on construction sites
- Personnel planning
- Marketing campaign planning
- Projects exploiting the CP method
  - Aids planning (on TF1)
  - Planning for « call-centers »

Source : Antoine Jeanjean

Other applications

More applications of the web-site

www.csplib.org

There are 88 applications !

Contents

Introduction

Applications of Constraint Programming

Constraint Satisfaction Problems

Modelling examples

Solving Constraint Satisfaction Problems

Constraint Satisfaction Problem (CSP)

CSP is a triple $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, where :

- **X** is the set of variables $\{x_1, \ldots, x_n\}$,
- **D** is the set of domains $\{D_{x_1}, \ldots, D_{x_n}\}$ (sets of possible values) for these variables,
- **C** is the set of constraints

$$\{C_i(x_{i_1}, \ldots, x_{n_i})\}_{i \in |C|}.$$

Every constraint C_i restricts the values that variables $\{x_{i_1}, \ldots, x_{i_n}\}$ can take simultaneously.

Domains

The domains can be

- finite sets :

$$\{1, 2, \ldots, n\}, \quad \{2, 3, 5\}, \quad \{\text{red}, \text{black}, \text{blue}\};$$

- intervals :

$$[0, k], \quad [1.2, 5.9];$$

- trees (not in this course).

Domains

The domains can be

- finite sets :

$$\{1, 2, \ldots, n\}, \quad \{2, 3, 5\}, \quad \{\text{red}, \text{black}, \text{blue}\};$$

- intervals :

$$[0, k], \quad [1.2, 5.9];$$

- trees (not in this course).

Domains

The domains can be

- finite sets :

$$\{1, 2, \ldots, n\}, \quad \{2, 3, 5\}, \quad \{\text{red}, \text{black}, \text{blue}\};$$

- intervals :

$$[0, k], \quad [1.2, 5.9];$$

- trees (not in this course).

Constraints

Constrains can be

- logic :

$$x = 1 \text{ or } y = 3, \quad x = 2 \Rightarrow y = 4;$$

- arithmetic :

$$x > y, \quad z = 2x + 3y - 5;$$

- explicit (tuples of possible values) :

$$(x, y) \in \{(0,0),(1,0),(2,2)\}, \quad (x,y,z) \in \{(1,2,3),(2,3,4)\};$$

- complex (global) :

$$\texttt{all} - \texttt{different}(x_1, \ldots, x_n).$$

Constraints

Constrains can be

- logic :

$$x = 1 \text{ or } y = 3, \quad x = 2 \Rightarrow y = 4;$$

- arithmetic :

$$x > y, \quad z = 2x + 3y - 5;$$

- explicit (tuples of possible values) :

$$(x, y) \in \{(0,0), (1,0), (2,2)\}, \quad (x, y, z) \in \{(1,2,3), (2,3,4)\};$$

- complex (global) :

$$\texttt{all} - \texttt{different}(x_1, \ldots, x_n).$$

Constraints

Constrains can be

- logic :

$$x = 1 \text{ or } y = 3, \quad x = 2 \Rightarrow y = 4;$$

- arithmetic :

$$x > y, \quad z = 2x + 3y - 5;$$

- explicit (tuples of possible values) :

$$(x, y) \in \{(0,0),(1,0),(2,2)\}, \quad (x, y, z) \in \{(1,2,3),(2,3,4)\};$$

- complex (global) :

$$\texttt{all} - \texttt{different}(x_1, \ldots, x_n).$$

Constraints

Constrains can be

- logic :

$$x = 1 \text{ or } y = 3, \quad x = 2 \Rightarrow y = 4;$$

- arithmetic :

$$x > y, \quad z = 2x + 3y - 5;$$

- explicit (tuples of possible values) :

$$(x, y) \in \{(0,0),(1,0),(2,2)\}, \quad (x, y, z) \in \{(1,2,3),(2,3,4)\};$$

- complex (global) :

$$\texttt{all} - \texttt{different}(x_1, \ldots, x_n).$$

Arity of constraints

Constraint can have an arbitrary arity :

- A constraint is unary if it contains one variable ($x = 4$)
- A constraint is binary if it contains two variables ($x + y = 9$)
- A constraint is *n*-ary if it contains *n* variables

The notion « *n*-ary » is used for a constraint such that the number of variables it contains is not known a priori (for example, $\texttt{all} - \texttt{different}$)

Solutions

Solution is an assignment of values $(v_1, \ldots, v_n)$ to variables $(x_1, \ldots, x_n)$ such that

- the values are in domains of variables : $v_j \in D_{x_j}, \forall j$;
- all constrains C_i are satisfied.

Solutions

Solution is an assignment of values $(v_1, \ldots, v_n)$ to variables $(x_1, \ldots, x_n)$ such that

- the values are in domains of variables : $v_j \in D_{x_j}, \forall j$;
- all constrains C_i are satisfied.

A CSP is satisfiable if it has a solution.

Solutions

Solution is an assignment of values $(v_1, \ldots, v_n)$ to variables $(x_1, \ldots, x_n)$ such that

- the values are in domains of variables : $v_j \in D_{x_j}, \forall j$;
- all constrains C_i are satisfied.

A CSP is satisfiable if it has a solution.

Solve a CSP $\Leftrightarrow$ determine if it is satisfiable or not.

Solutions

Solution is an assignment of values $(v_1, \ldots, v_n)$ to variables $(x_1, \ldots, x_n)$ such that

- the values are in domains of variables : $v_j \in D_{x_j}, \forall j$;
- all constrains C_i are satisfied.

An example

- Variables : x, y and z.
- Domains : $D_x = D_y = D_z = \{1, 2, 3\}$.
- One constraint : $x + y = z$.
- Solutions : $(1, 1, 2)$, $(1, 2, 3)$, $(2, 1, 3)$.

Solutions

Solution is an assignment of values $(v_1, \ldots, v_n)$ to variables $(x_1, \ldots, x_n)$ such that

- the values are in domains of variables : $v_j \in D_{x_j}, \forall j$;
- all constrains C_i are satisfied.

An example

- Variables : x, y and z.
- Domains : $D_x = D_y = D_z = \{1, 2, 3\}$.
- One constraint : $x + y = z$.
- Solutions : $(1, 1, 2)$, $(1, 2, 3)$, $(2, 1, 3)$.

Problem types in CP

- Find a solution, if one exists (classic).
- Find all solutions.
- Find a solution which minimizes of maximizes a criterion (solved using dichotomy).

Problem types in CP

- Find a solution, if one exists (classic).
- Find all solutions.
- Find a solution which minimizes of maximizes a criterion (solved using dichotomy).

Problem types in CP

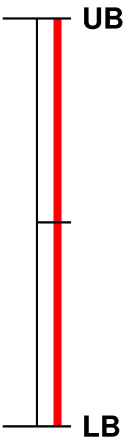
- Find a solution, if one exists (classic).
- Find all solutions.
- Find a solution which minimizes of maximizes a criterion (solved using dichotomy).

Dichotomy

General algorithm (minimisation)

Find a lower bound (**LB**) and upper bound (**UB**) for the value of the objective function;
while UB $-$ **LB** *is large* **do**
  test $\leftarrow$ **LB** $+ \frac{\mathbf{UB}-\mathbf{LB}}{2}$;
  if *exists a solution* $\leq$ test **then**
    UB $\leftarrow$ test;
    save this solution;
  else
    LB $\leftarrow$ test;

Example

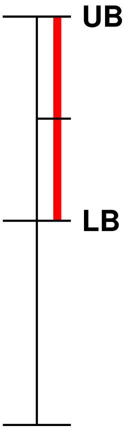


Dichotomy

General algorithm (minimisation)

Find a lower bound (**LB**) and upper bound (**UB**) for the value of the objective function;
while UB $-$ **LB** *is large* **do**
    $\texttt{test} \leftarrow \mathbf{LB} + \frac{\mathbf{UB}-\mathbf{LB}}{2}$;
    if *exists a solution* $\leq \texttt{test}$ **then**
        UB $\leftarrow \texttt{test}$;
        save this solution;
    else
        LB $\leftarrow \texttt{test}$;

Example

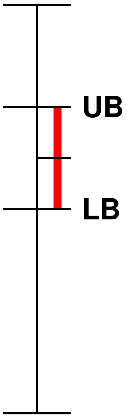


Dichotomy

General algorithm (minimisation)

Find a lower bound (**LB**) and upper bound (**UB**) for the value of the objective function;
while **UB** $-$ **LB** *is large* **do**
  `test` $\leftarrow$ **LB** $+ \frac{\textbf{UB}-\textbf{LB}}{2}$;
  if *exists a solution* $\leq$ `test` **then**
    UB $\leftarrow$ `test`;
    save this solution;
  else
    LB $\leftarrow$ `test`;

Example

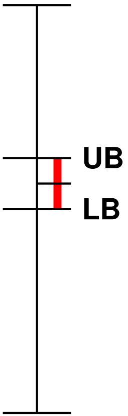


Dichotomy

General algorithm (minimisation)

Find a lower bound (**LB**) and upper bound (**UB**) for the value of the objective function;
while UB − LB *is large* **do**
    `test` $\leftarrow$ **LB** + $\frac{\textbf{UB}-\textbf{LB}}{2}$;
    if *exists a solution* $\leq$ `test` **then**
        UB $\leftarrow$ `test`;
        save this solution;
    else
        LB $\leftarrow$ `test`;

Example



Contents

Introduction

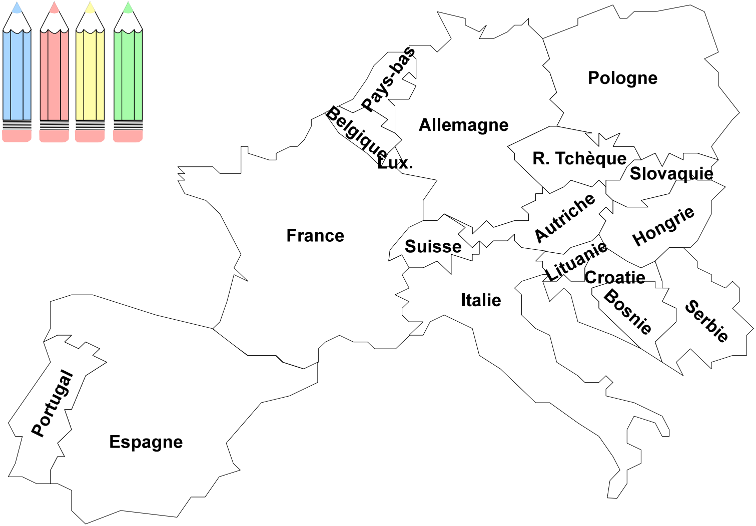
Applications of Constraint Programming

Constraint Satisfaction Problems

Modelling examples

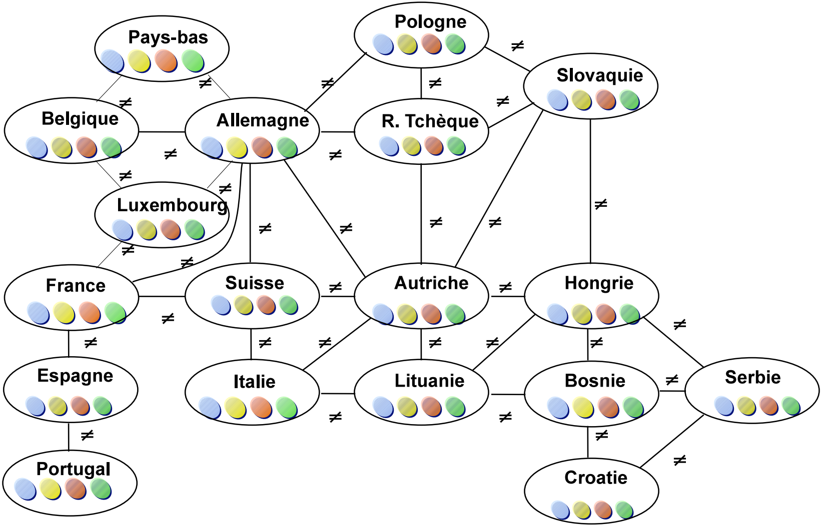
Solving Constraint Satisfaction Problems

Example I — Map coloring



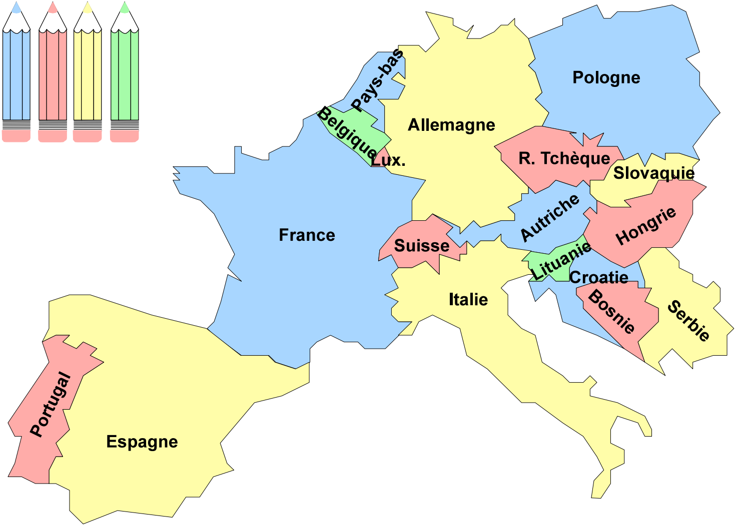
Source : Philippe Baptiste

Example I — Map coloring



Source : Philippe Baptiste

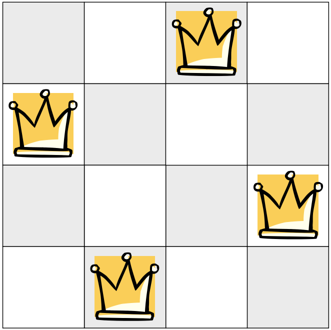
Example I — Map coloring



Source : Philippe Baptiste

Example II — N queens

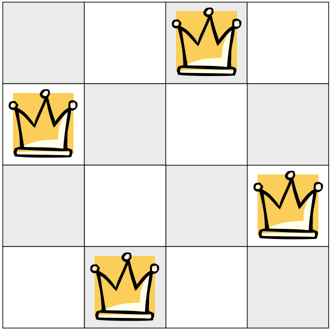
Given a chessboard with $N \times N$ cells, put N queens in such a way that no queen is able to capture another one.



- Variables :
- Domains :
- Constraints :

Example II — N queens

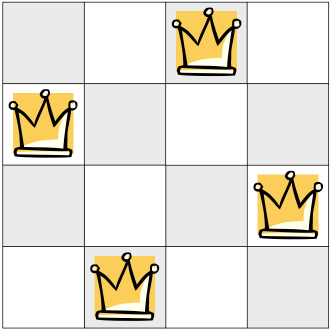
Given a chessboard with $N \times N$ cells, put N queens in such a way that no queen is able to capture another one.



- Variables : x_i — position of the queen in column i.
- Domains :
- Constraints :

Example II — N queens

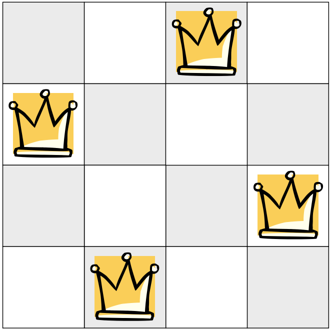
Given a chessboard with $N \times N$ cells, put N queens in such a way that no queen is able to capture another one.



- **Variables** : x_i — position of the queen in column i.
- **Domains** : $D_{x_i} = \{1, \ldots, N\}, \forall i$.
- **Constraints** :

Example II — N queens

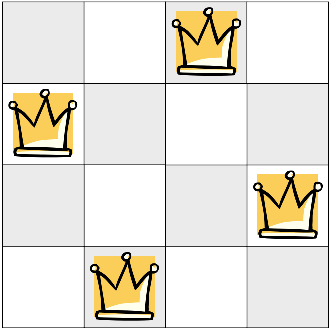
Given a chessboard with $N \times N$ cells, put N queens in such a way that no queen is able to capture another one.



- **Variables** : x_i — position of the queen in column i.
- **Domains** : $D_{x_i} = \{1, \ldots, N\}, \forall i$.
- **Constraints** :
  - $x_i \neq x_j, \forall i, j, 1 \leq i < j \leq N$,
  - $x_i \neq x_j + (j - i), 1 \leq i < j \leq N$,
  - $x_i \neq x_j + (i - j), 1 \leq i < j \leq N$.

Example II — N queens

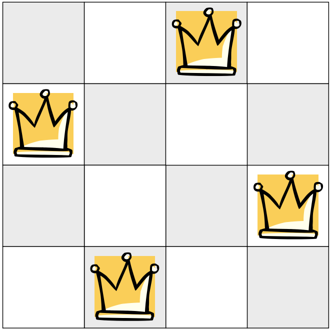
Given a chessboard with $N \times N$ cells, put N queens in such a way that no queen is able to capture another one.



- **Variables** : x_i — position of the queen in column i.
- **Domains** : $D_{x_i} = \{1, \ldots, N\}, \forall i$.
- **Constraints** :
  - $x_i \neq x_j, \forall i, j, 1 \leq i < j \leq N$,
  - $x_i \neq x_j + (j - i), 1 \leq i < j \leq N$,
  - $x_i \neq x_j + (i - j), 1 \leq i < j \leq N$.

Example II — N queens

Given a chessboard with $N \times N$ cells, put N queens in such a way that no queen is able to capture another one.



- Variables : x_i — position of the queen in column i.
- Domains : $D_{x_i} = \{1, \ldots, N\}, \forall i$.
- Constraints :
  - `all-different`$(x_1, \ldots, x_N)$,
  - $x_i \neq x_j + (j - i), 1 \leq i < j \leq N$,
  - $x_i \neq x_j + (i - j), 1 \leq i < j \leq N$.

Example III — Sudoku

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

- Variables :
- Domains :

Constraints :

Example III — Sudoku

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

- Variables : x_{ij} — digit in cell (i, j).
- Domains :

Constraints :

Example III — Sudoku

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

- Variables : x_{ij} — digit in cell (i, j).
- Domains : $D_{x_{ij}} = \{1, \ldots, 9\}, \forall (i, j)$.

Constraints :

Example III — Sudoku

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

- **Variables** : x_{ij} — digit in cell (i, j).
- **Domains** : $D_{x_{ij}} = \{1, \ldots, 9\}, \forall (i, j)$.

Constraints :

- The digits in each line are different :
  `all-different`$(x_{i1}, x_{i2}, \ldots, x_{i9})$, $1 \leq i \leq 9$,
- The digits in each column are different :
  `all-different`$(x_{1j}, x_{2j}, \ldots, x_{9j})$, $1 \leq j \leq 9$,
- The digits in each block 3×3 are different :
  `all-different`$(x_{3k+1,3l+1}, x_{3k+1,3l+2}, \ldots, x_{3k+3,3l+3})$,
  $0 \leq k, l \leq 2$.

Example III — Sudoku

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

- Variables : x_{ij} — digit in cell (i, j).
- Domains : $D_{x_{ij}} = \{1, \ldots, 9\}$, $\forall (i, j)$.

Constraints :

- The digits in each line are different :
  `all-different`$(x_{i1}, x_{i2}, \ldots, x_{i9})$, $1 \le i \le 9$,
- The digits in each column are different :
  `all-different`$(x_{1j}, x_{2j}, \ldots, x_{9j})$, $1 \le j \le 9$,
- The digits in each block 3×3 are different :
  `all-different`$(x_{3k+1,3l+1}, x_{3k+1,3l+2}, \ldots, x_{3k+3,3l+3})$,
  $0 \le k, l \le 2$.

Example III — Sudoku

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

- Variables : x_{ij} — digit in cell (i, j).
- Domains : $D_{x_{ij}} = \{1, \ldots, 9\}, \forall (i, j)$.

Constraints :

- The digits in each line are different :
  `all-different`$(x_{i1}, x_{i2}, \ldots, x_{i9})$, $1 \leq i \leq 9$,
- The digits in each column are different :
  `all-different`$(x_{1j}, x_{2j}, \ldots, x_{9j})$, $1 \leq j \leq 9$,
- The digits in each block 3×3 are different :
  `all-different`$(x_{3k+1,3l+1}, x_{3k+1,3l+2}, \ldots, x_{3k+3,3l+3})$,
  $0 \leq k, l \leq 2$.

Example IV - Giving a change

We are interested in modelling a vending machine. A user inserts coins for a total value of T eurocents, then he selects a drink for the price of P eurocents. We need to calculate the change to give, knowing that the machine has E_2 coins of 2€, E_1 coins of 1€, C_{50} coins of 50 eurocents, C_{20} coins of 20 eurocents, and C_{10} coins of 10 eurocents.

- Variables : $x_{E2}, x_{E1}, x_{C50}, x_{C20}, x_{C10}$.
- Domains : $D_{x_{E2}} = \{0, 1, \ldots, E_2\}$, $D_{x_{E1}} = \{0, 1, \ldots, E_1\}$,...
- Constraint :

$$200x_{E2} + 100x_{E1} + 50x_{C50} + 20x_{C20} + 10x_{C10} = T - P$$

- If we want to minimize a number of coins to give, we need to specify the objective function :

$$\min x_{E2} + x_{E1} + x_{C50} + x_{C20} + x_{C10}$$

Example IV - Giving a change

We are interested in modelling a vending machine. A user inserts coins for a total value of T eurocents, then he selects a drink for the price of P eurocents. We need to calculate the change to give, knowing that the machine has E_2 coins of 2€, E_1 coins of 1€, C_{50} coins of 50 eurocents, C_{20} coins of 20 eurocents, and C_{10} coins of 10 eurocents.

- Variables : $x_{E2}, x_{E1}, x_{C50}, x_{C20}, x_{C10}$.
- Domains : $D_{x_{E2}} = \{0, 1, \ldots, E_2\}$, $D_{x_{E1}} = \{0, 1, \ldots, E_1\}$,...
- Constraint :

$$200x_{E2} + 100x_{E1} + 50x_{C50} + 20x_{C20} + 10x_{C10} = T - P$$

- If we want to minimize a number of coins to give, we need to specify the objective function :

$$\min x_{E2} + x_{E1} + x_{C50} + x_{C20} + x_{C10}$$

Example IV - Giving a change

We are interested in modelling a vending machine. A user inserts coins for a total value of T eurocents, then he selects a drink for the price of P eurocents. We need to calculate the change to give, knowing that the machine has E_2 coins of 2€, E_1 coins of 1€, C_{50} coins of 50 eurocents, C_{20} coins of 20 eurocents, and C_{10} coins of 10 eurocents.

- Variables : $x_{E2}, x_{E1}, x_{C50}, x_{C20}, x_{C10}$.
- Domains : $D_{x_{E2}} = \{0, 1, \ldots, E_2\}$, $D_{x_{E1}} = \{0, 1, \ldots, E_1\}$,...
- Constraint :

$$200x_{E2} + 100x_{E1} + 50x_{C50} + 20x_{C20} + 10x_{C10} = T - P$$

- If we want to minimize a number of coins to give, we need to specify the objective function :

$$\min x_{E2} + x_{E1} + x_{C50} + x_{C20} + x_{C10}$$

Example IV - Giving a change

We are interested in modelling a vending machine. A user inserts coins for a total value of T eurocents, then he selects a drink for the price of P eurocents. We need to calculate the change to give, knowing that the machine has E_2 coins of 2€, E_1 coins of 1€, C_{50} coins of 50 eurocents, C_{20} coins of 20 eurocents, and C_{10} coins of 10 eurocents.

- Variables : $x_{E2}, x_{E1}, x_{C50}, x_{C20}, x_{C10}$.
- Domains : $D_{x_{E2}} = \{0, 1, \ldots, E_2\}$, $D_{x_{E1}} = \{0, 1, \ldots, E_1\}$,...
- Constraint :

$$200x_{E2} + 100x_{E1} + 50x_{C50} + 20x_{C20} + 10x_{C10} = T - P$$

- If we want to minimize a number of coins to give, we need to specify the objective function :

$$\min x_{E2} + x_{E1} + x_{C50} + x_{C20} + x_{C10}$$

Contents

Introduction

Applications of Constraint Programming

Constraint Satisfaction Problems

Modelling examples

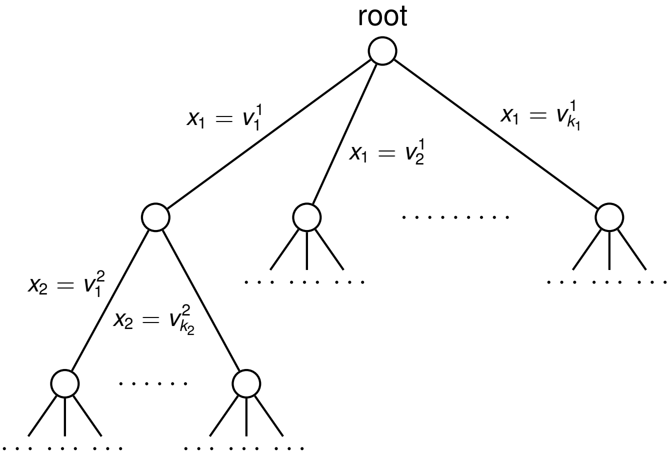
Solving Constraint Satisfaction Problems

Main idea

The method of Constraint Programming (which solves a CSP) is based on working with partial solutions and enumeration tree :

- We assign a value to a variable and see if all constraints are still satisfied.
- If not, we « backtrack » and try another value.
- To avoid complete enumeration, each time a variable takes a value, incompatible (with this decision) variables are removed (this process called propagation).

Enumeration tree



An example of simple propagation

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

$$D = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

An example of simple propagation

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

$$D = \{1, 2, \cancel{3}, 4, 5, \cancel{6}, 7, \cancel{8}, \cancel{9}\}$$

An example of simple propagation

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

$$D = \{1, 2, \cancel{3}, \cancel{4}, \cancel{5}, \cancel{6}, \cancel{7}, \cancel{8}, \cancel{9}\}$$

An example of simple propagation

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

$$D = \{1, \cancel{2}, \cancel{3}, \cancel{4}, \cancel{5}, \cancel{6}, \cancel{7}, \cancel{8}, \cancel{9}\}$$

An example of advanced propagation

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

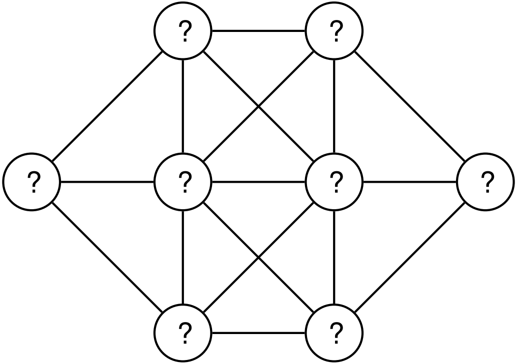
An example of advanced propagation

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

An example of advanced propagation

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

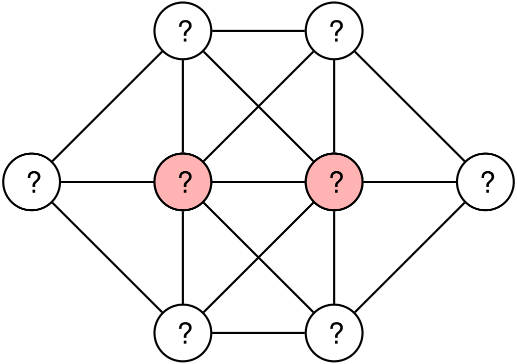
An example of complete solution



Problem : assign values from 1 to 8 to vertices, each value should appear once, consecutive values should not be assigned to adjacent vertices

Source : Patrick Prosser

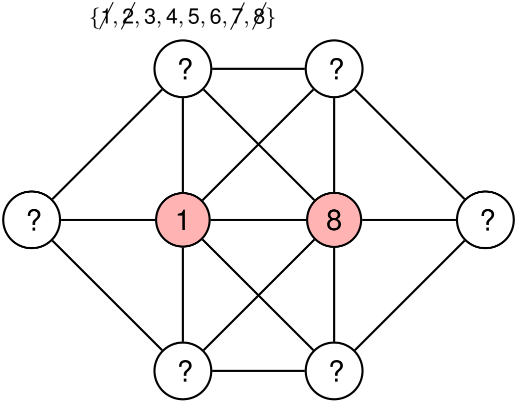
An example of complete solution



Be ready to do a backtrack.
Which vertices are more difficult to enumerate ?
Which values are less restraining ?

Source : Patrick Prosser

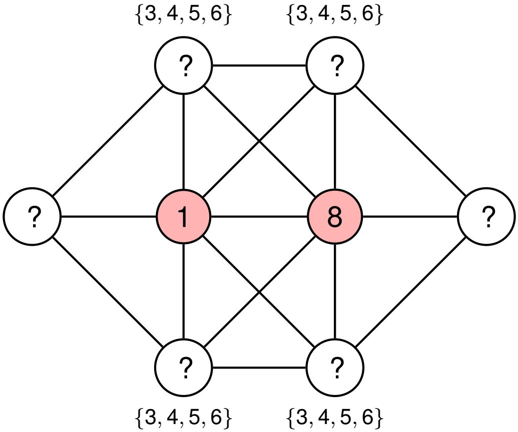
An example of complete solution



We can now remove several variables from the domains of other vertices.

Source : Patrick Prosser

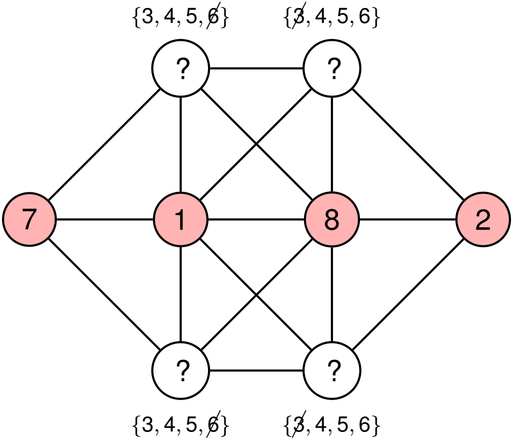
An example of complete solution



We can now remove several variables from the domains of other vertices.

Source : Patrick Prosser

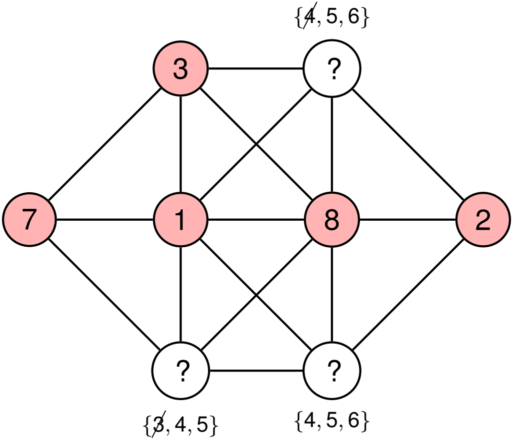
An example of complete solution



We can now remove several variables from the domains of other vertices.

Source : Patrick Prosser

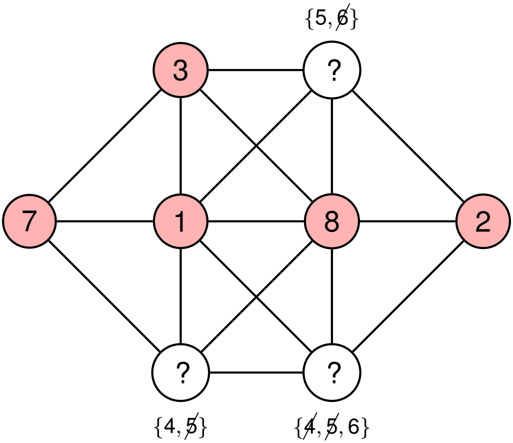
An example of complete solution



We guess now a value for a vertex.
Be ready to do a backtrack.

Source : Patrick Prosser

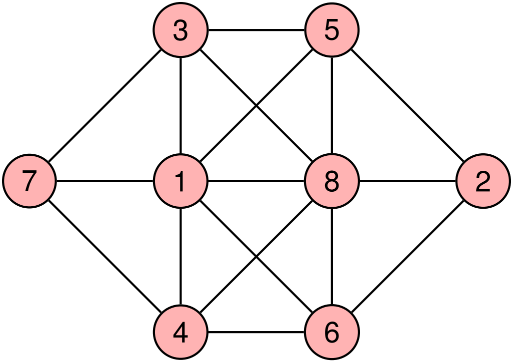
An example of complete solution



We propagate this decision.

Source : Patrick Prosser

An example of complete solution

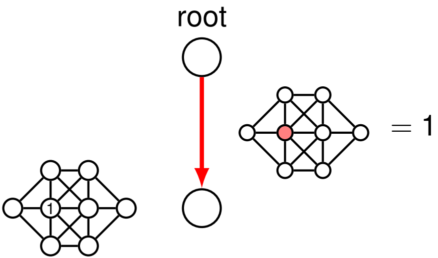


A solution.

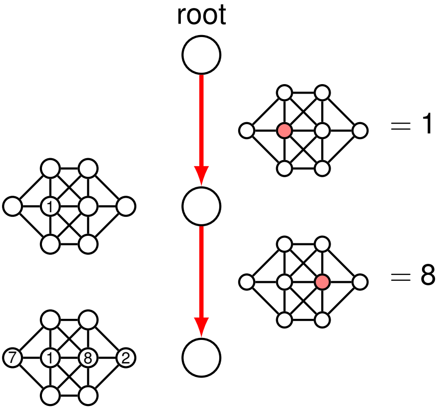
Source : Patrick Prosser

Example : enumeration tree

Example : enumeration tree



Example : enumeration tree



Example : enumeration tree