HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Introduction to Constraint Programming

Almost slides from the course CP given by
Pascal Van Hentenryck & Yves Deville

ONE LOVE. ONE FUTURE.

Content

- A first challenge
- Constraint Satisfaction Problem
- Constraint Programming
- Examples with Or-Tools

Sudoku

- Fill the empty cells with numbers in {1,..,9}
- **Constraint** : each number only appears once in each
  - Row
  - Column
  - Region

easy

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

hard

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

Sudoku

easy

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

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

hard

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

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

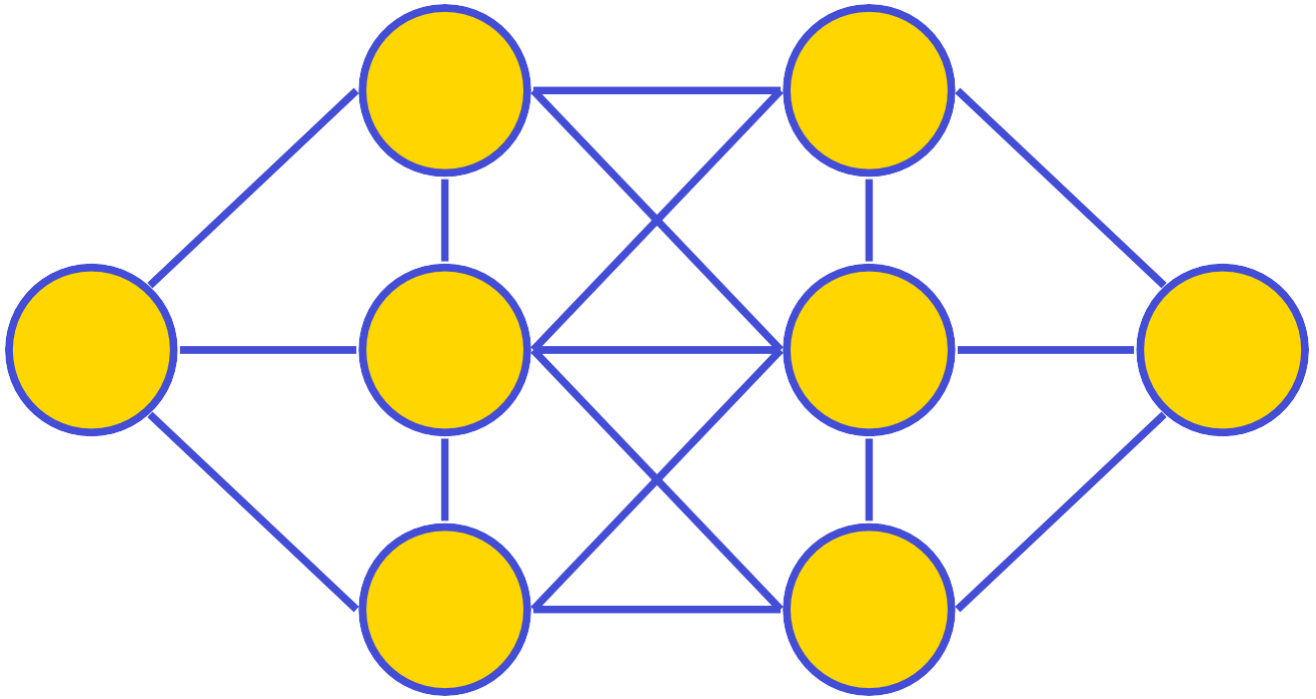
Sudoku

- Sudoku is a (very) simple example of Constraint Satisfaction Problem (CSP)
  - A set of variables, defined over domains
  - A set of constraints over the variables
- What is a solution?
  - A solution is an assignment of values to the variables
  - does not violate any constraint

A challenge from T. Walsh

- Place numbers 1 through 8 on nodes
  - Each number appears exactly once
  - No connected nodes have consecutive number

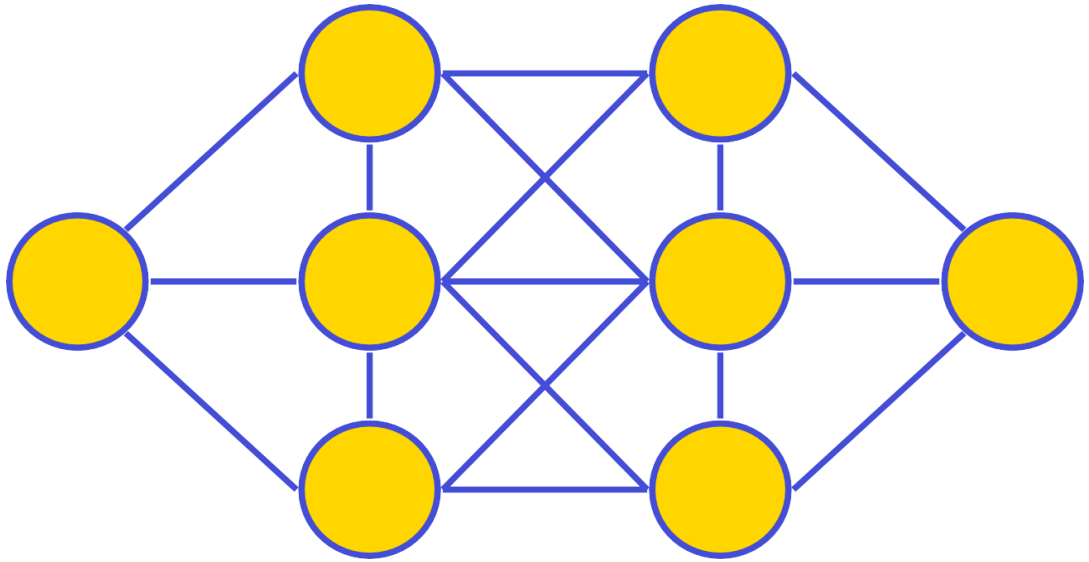
How long
will you take ?



- This is an example of a Constraint Satisfaction Problem (CSP)
  - A set of variables defined over domains
  - A set of constraints over the variables
- It can be solved by Constraint Programming

How did you solved this problem manually ?

Which techniques did you used ?



Content

- A first challenge
- **Constraint Satisfaction Problem**
- Constraint Programming
- Examples with Or-Tools

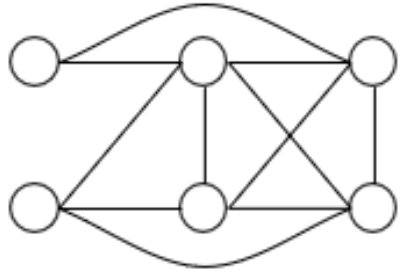
Constraint Optimization Problem

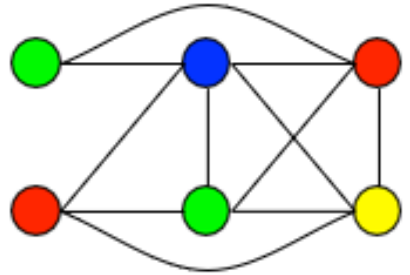
- Constraint Satisfaction Problem (CSP)
  - A set of variables, defined over finite domains
  - A set of constraints over the variables
- Solution
  - A solution is a assignment of values to the variables
  - No violation of constraint
- Constraint Optimization Problem (COP)
  - Add an objective function
  - Find a solution maximizing the objective function

Many CSP / COP are complex and challenging (NP-hard)

A simple example

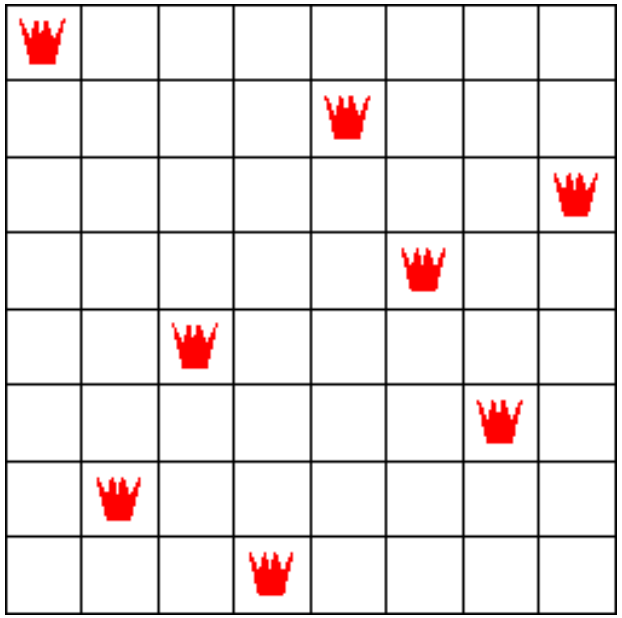
- **CSP** : Graph Coloring with 4 colors
  - Graph G=(V,E)
  - Decision variables
    - color[v] v < V
  - Domain of variables : {red,blue,green,yellow}
  - Constraints
    - 6(v,w) < E : **color[v] ≠ color[w]**
- **COP** : minimize the used colors

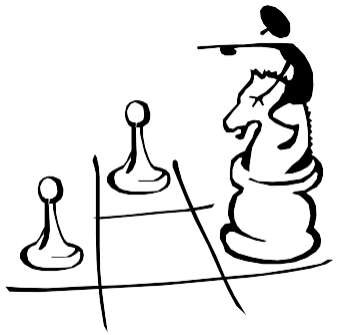




Example : N-queens

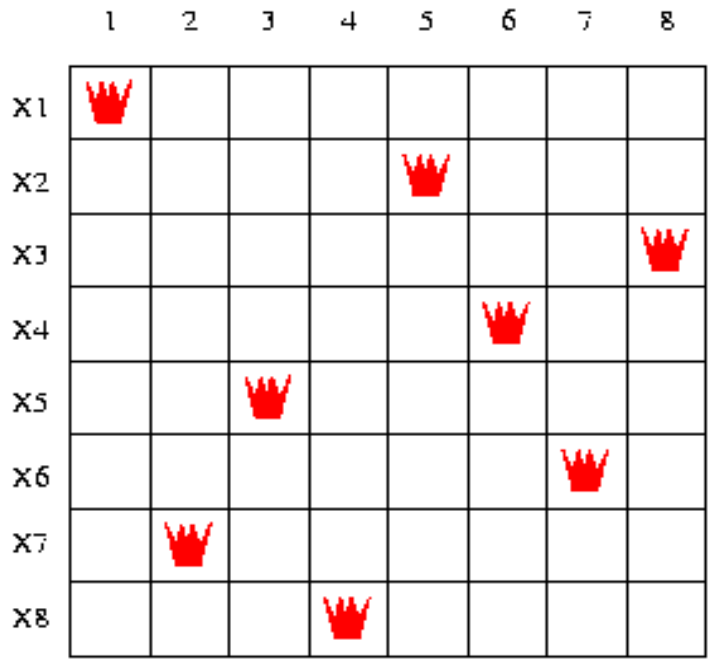
- How to place 8 queens on a 8x8 board such that they do not attack each other
- A solution





N-queens as a CSP

- **Variables**
  - X_i : position (column) of the queen on row i ($1 \leq i \leq 8$)
- **Domains**
  - Domain of X_i :
    - {1,2,3,4,5,6,7,8}



N-queens as a CSP

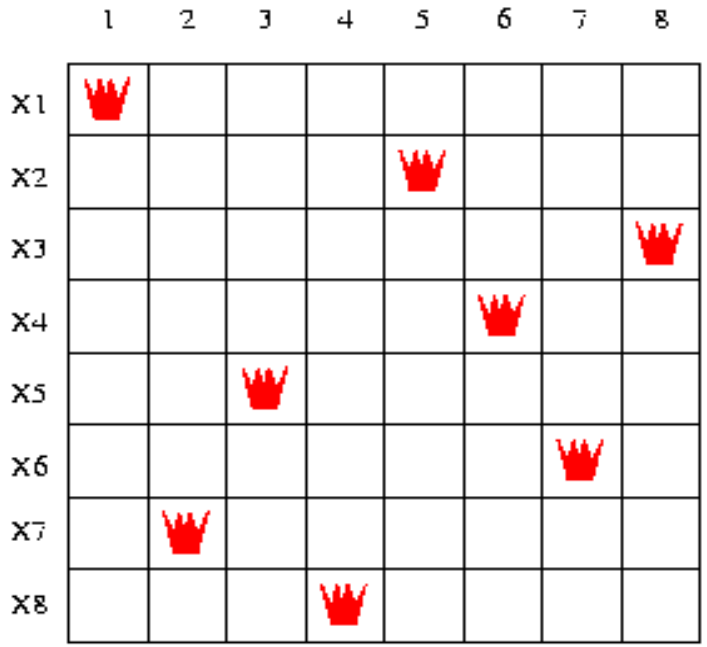
- **Constraints**
  - Two queens cannot be on the same column
    - $X_i \neq X_j$
  - Two queens cannot be on the same diagonal
    - $X_i - X_j \neq i - j$ $X_i - X_j \neq j - i$
- 8-queens
  - 8 variables
  - 84 contraints

X1≠X2, X1≠X3, X1≠X4, X1≠X5, X2≠X3, X2≠X4, X2≠X5, X3≠X4, X3≠X5, X4≠X5, …

X1-X2≠-1, X1-X2≠1, X1-X3≠-2, X1-X3≠2, X1-X4≠-3, X1-X4≠3, X1-X5≠-4, X1-X5≠4, X2-X3≠-1, X2-X3≠1, X2-X4≠-2, X2-X4≠2, X2-X5≠-3, X2-X5≠3, X3-X4≠-1, X3-X4≠1, X2-X5≠-2, X3-X5≠2, X4-X5≠-1, X4-X5≠1, …

N-queens as a CSP

- X_i : position of the queen on row i
  - $X_1 = 1$
  - $X_2 = 5$
  - $X_3 = 8$
  - $X_4 = 6$
  - $X_5 = 3$
  - $X_6 = 7$
  - $X_7 = 2$
  - $X_8 = 4$



Constraints

- Constraints as basic computational elements
  - $2X + 3Y \leq 17$ $\quad X^2 - Y^3 = 17$
- Constraints appear naturally in :
  - artificial intelligence
  - operational research
  - combinatorial optimization
  - graphics, visualisation
  - concurrency
  - databases
  - hardware design, verification
  - ...

Who does optimization ?

- Microsoft (1st largest software company)
- SAP (2nd largest software company)
- Oracle (3rd largest software company)
  - Supply chain management
- IBM (of course)
- Siebel, PeopleSoft
  - Personal management
- UPS
  - Dispatching
- Google, IBM, Carmen systems, Fair Isaac

Complexity of CSP

- Computational Complexity
  - NP-Hard or worse (at least exponential)
  - No reasonable approximation
- *Yet, it is important to solve them*
- The best specific algorithms
  - tough to design and implement
  - experimental

Solving Paradigms

- **Mathematical Programming** : *IP, MIP, column generation, SDP, ...*
- **Dynamic Programming**
- **Local Search** : tabu search, simulated annealing, genetic algorithms, evolutionary algorithms...
- Greedy algorithms, Ant Colony Optimization, greedy randomized, ...
- **Constraint Programming**
- ...

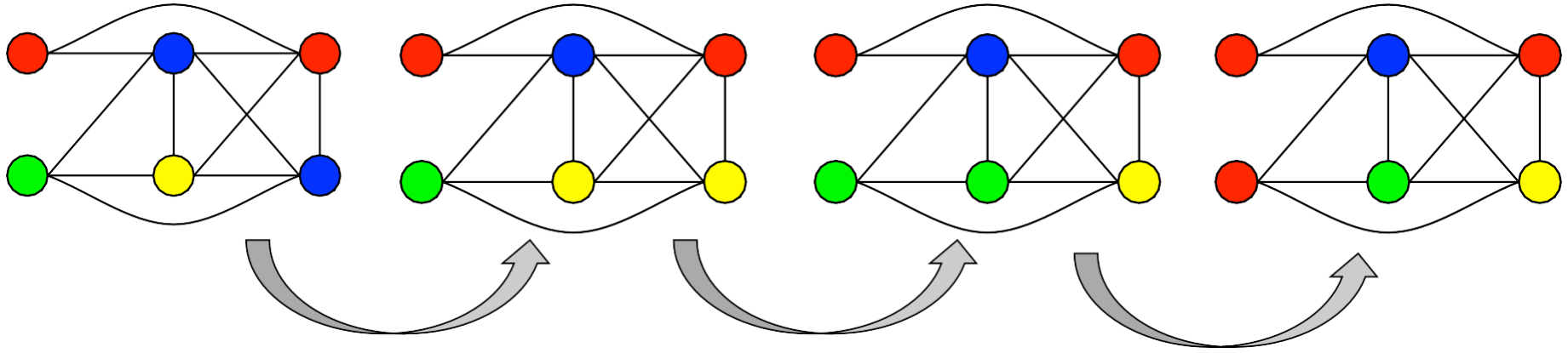
Solving Paradigms

Paradigms can usually be classified according to two criteria

- **Perturbative** vs **constructive**
- **Systematic** vs **incomplete**

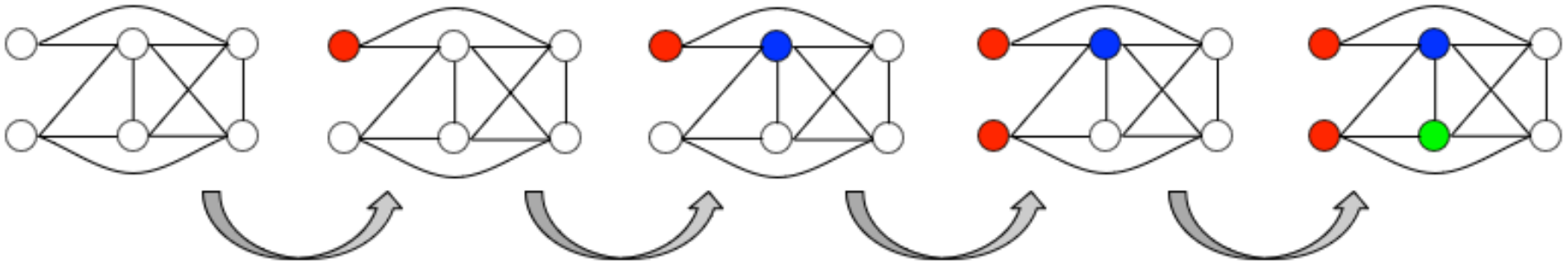
Perturbative vs Constructive

- **Perturbative** approaches
  - A candidate solution is a complete solution (a value is assigned to each variable)
  - Generation of a new solution by modifications, pertubation of an existing one
  - Search is performed on the succession of candidate solutions



Perturbative vs Constructive

- **Constructive** approaches
  - A candidate solutions is partial (some variables are not assigned)
  - A partial candidate solution is progressively extended, constructed



Systematic vs Incomplete

- **Systematic** approaches
  - Systematic exploration of the search space
  - Completeness
    - CSP : find an *exact* solution
    - COP : find the *best* solution, (+ proof of optimality)
    - If no solution, proof of non existence
  - Computationally expensive

Systematic vs Incomplete

- **Incomplete** approaches
  - Exploration based on a neighborhood function
  - Incomplete
    - CSP: find an *approximation* of a solution
    - COP: find a *good* solution
    - No guarrantee to find the (best) solution
  - Computationally less expensive

Paradigms

	Systematic	Incomplete
Pertubative	▪ Simplex ▪ Systematic local search	▪ **Local search** ▪ Evolutionary algo.
Constructive	▪ Dynamic Programming ▪ Branch & Bound ▪ **Constraint Programming**	▪ Greedy ▪ Ant Colony Optimizatio ▪ GRASP

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - **What is CP ?**
  - Constraints and propagation
  - Modeling
  - Searching
  - Global constraint
  - Computation domain
- Examples with Or-Tools

Constraint programming

- A field at the intersection of
  - Artificial intelligence
  - Operational research
- A framework for solving CSP

Constraint-Based Modelling

Solution process
=
Model + Search

- The model
  - what the solutions are and their quality
- The search
  - how to find the solutions

CP and CBLS

- **Constraint Programming**
  - *Modeling with Constraints*
  - Systematic and constructive paradigm
  - Computation based on **branch & propagate**
- **Constraint-Based Local Search**
  - *Modeling with Constraints*
  - Incomplete and perturbative paradigm
  - Computation based on **neigborhood**

The CP computation model

- Iterations of two steps
- **Propagation**
  - reducing the domains of the variables
  - uses constraints to detect impossible values of variables
  - solve a relaxation of the problem
- **Branch** (also called **Search**)
  - Decompose into subproblems
  - nondeterministic choices (e.g., giving a value for a variable)
  - automatic support for backtracking

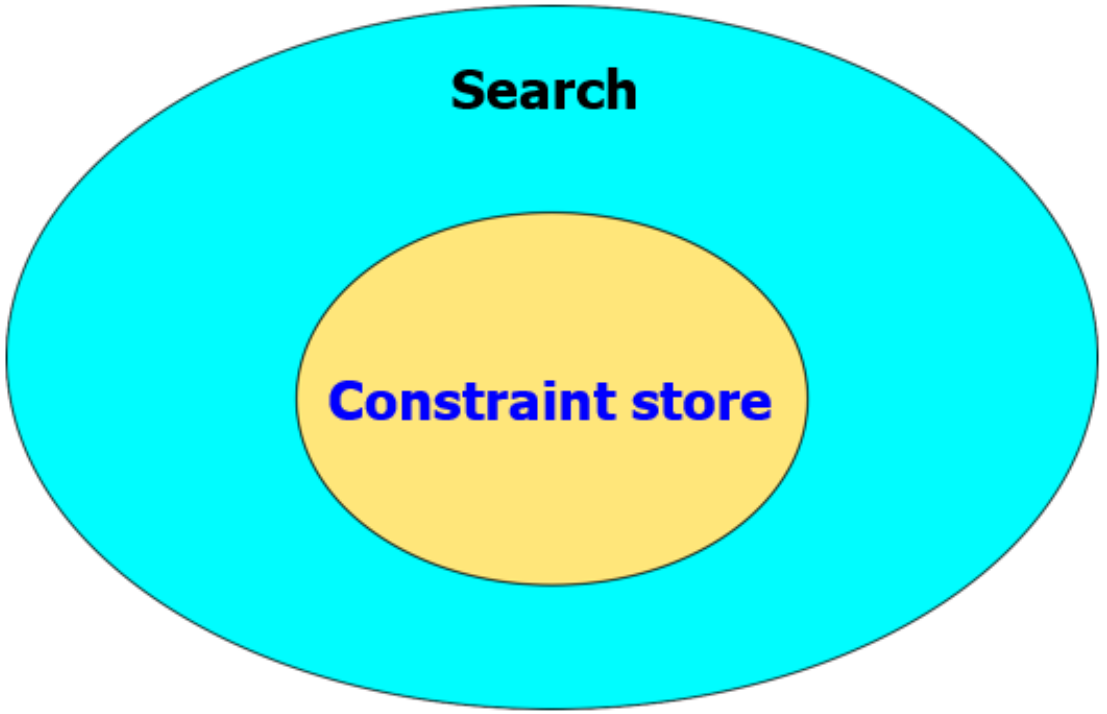
Constraint Programming

- Began in 1980s from AI world
  - Prolog III (Marseilles, France)
  - CLP(R) (Melbourne & IBM)
  - CHIP (ECRC, Germany)
- Active research area
  - Specialized conferences (CP, CP-AI-OR, …)
  - Journal (Constraints, CP Letters)
  - Commercial products

The CP Language

- A rich constraint language
  - Arithmetic, higher-order, logical constraints
  - Global constraints for natural substructures
- Specification of a search procedure
  - Definition of search tree to explore
  - Specification of exploration strategy
- Separation of concerns
  - Constraints and search are separated

Computational Model



Domains of computation

Constraint propagation techniques are related to specific **computation domains**

- Finite domains
- Finite sets
- Boolean
- Interval
- Reals
- Graphs
- …

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - What is CP ?
  - **Constraints and propagation**
  - Modeling
  - Searching
  - Global constraint
  - Computation domain
- Examples with Or-Tools

What is a constraint ?

- Numerical inequalities and equations
- Combinatorial / global constraints
  - natural subproblems arising of many applications
  - e.g. a set of activities A cannot overlap in time
- Logical/threshold combinations of these
- …

Constraints

- Powerful modelling tool
  - Numerical constraints
    - $x = y + 2z$, $x \neq y - 2$, $x \leq y - z$, $|x-y| \neq 2$, ...
  - Symbolic Constraints
    - $cost = \sum_{i<R} price[rack[i]]$
  - Composed constraints
    - $\sum_{1 \leq i \leq k} (x[i]=y) \leq k$ (at most k occurences of y in x[])
    - $load[j] = \sum_{i<Obj} (bin[i]=j).w[i]$ (load of bin j in bin packing)
  - Global constraints
    - alldifferent(x[1..n])
    - knapsack(x[1..n], weigth[1..n], cap)
    - Grammar constraints, scheduling constraints, ...

Propagation

Why propagation ?

- Exhaustive search of a solution: exponential time

Objectives of propagation

- Reduce the search space
- Find an equivalent CSP to the original one with *smaller domains* of variables

Techniques

- Consider the constraints *locally*
- **Consistency techniques**
  - Domain Consistency : consider each possible value in the domain
  - Bound Consistency : only considers the bounds of the domains
  - Other consistencies

Pruning

- Domain Consistency
  - The holy grail
- A constraint is **domain-consistent** if,
  - for every variable x and every value in the domain of x,
  - there exist values in the domains of the other variables
  - such that the constraint is satisfied for these values

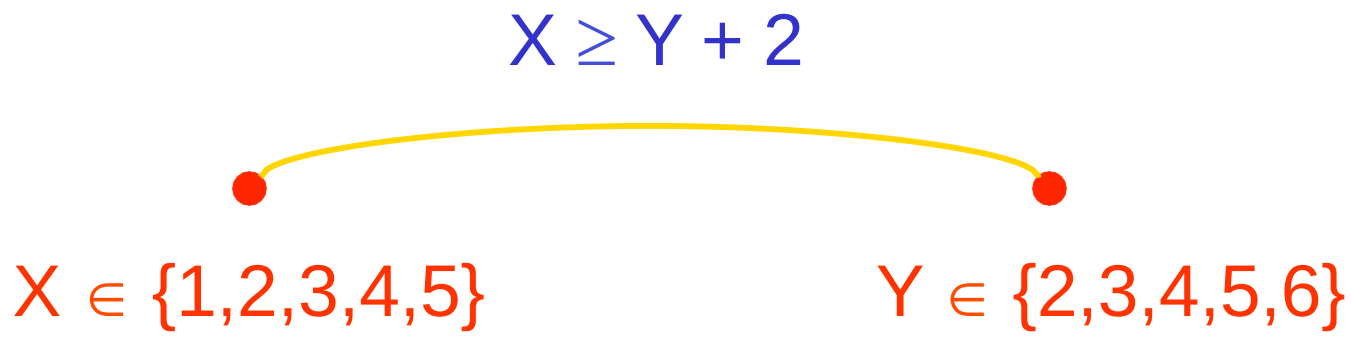
Domain consistency

$$X \geq Y + 2$$

$X \in \{1,2,3,4,5\}$ $Y \in \{2,3,4,5,6\}$

- remove any values from the domain of X for which there is no value in the domain of Y that satisfies the constraint (and vice versa)

Domain consistency



- remove any values from the domain of X for which there is no value in the domain of Y that satisfies the constraint (and vice versa)

Domain consistency

$$X \geq Y + 2$$

$X \in \{4,5\}$ $Y \in \{2,3\}$

- This constraint is now domain consistent
- Possible objective : make *all* constraints domain consistent
- Should be done *efficiently*

Consistency does not solve the CSP !

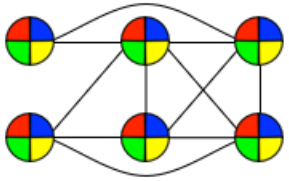
Bound Consistency

- Achieving domain consistency can be computationally expensive
- Bound consistency is a weaker form of consistency
  - a domain is approximated by an interval [min D(x), max D(x)]
  - only verify that the bounds of the domain are in a solution of the constraint

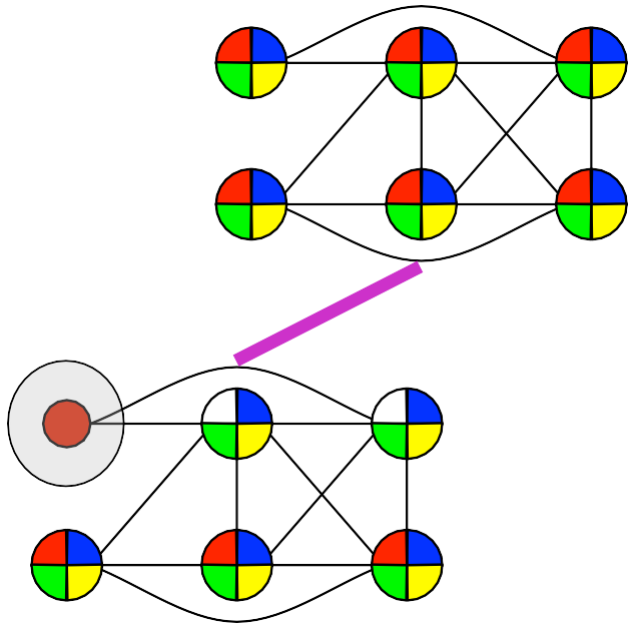
Bound Consistency

- Specialized to each constraint type
  - x + y + z = 10
  - x in {0,1,2,3,4}, y in {1,2,4}, z in {1,2,4}
- Domain consistency (AC) gives
  - x in {2,4}, y in {2,4}, z in {2,4}
  - expensive to compute ($O(d^3)$)
- Bound consistency (BC) gives
  - x in in {2,3,4}, y in {2,4}, z in {2,4}
  - can be computed very efficiently ($O(1)$)

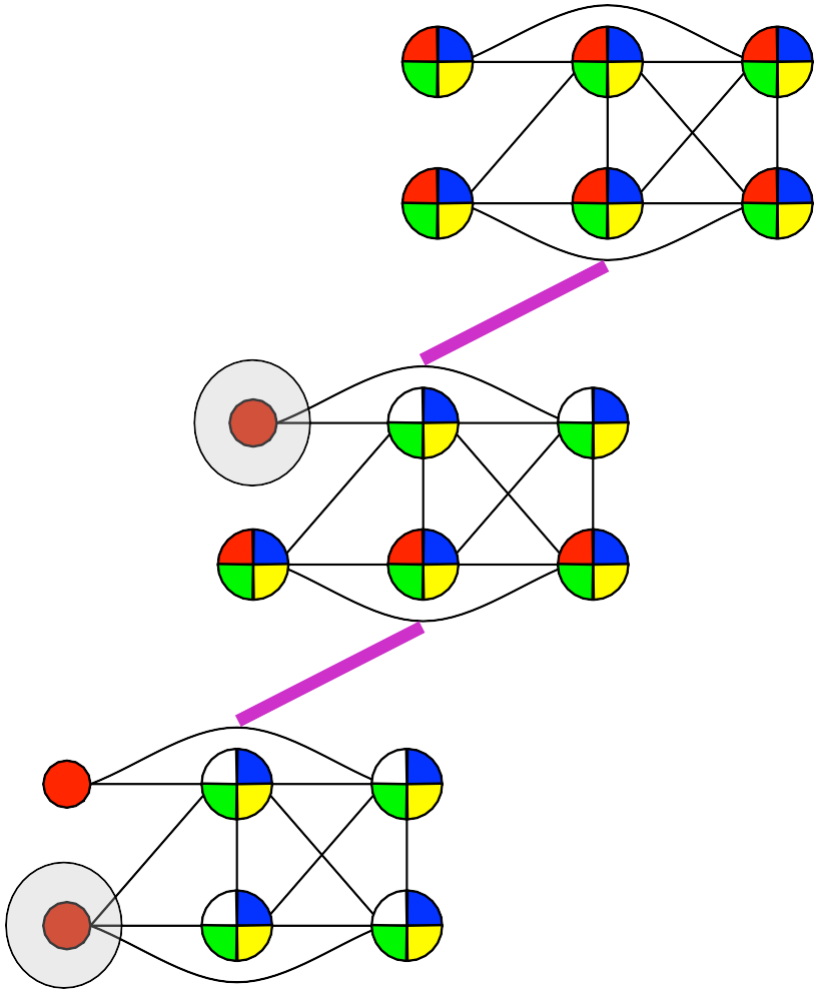
Graph Coloring CP



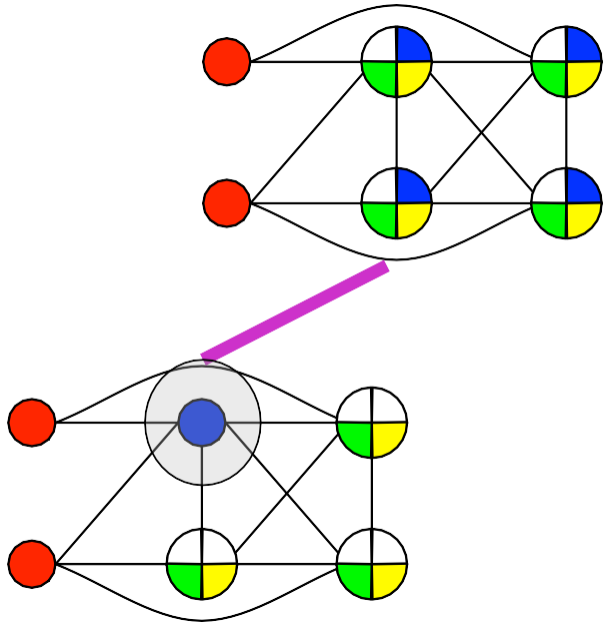
Graph Coloring CP



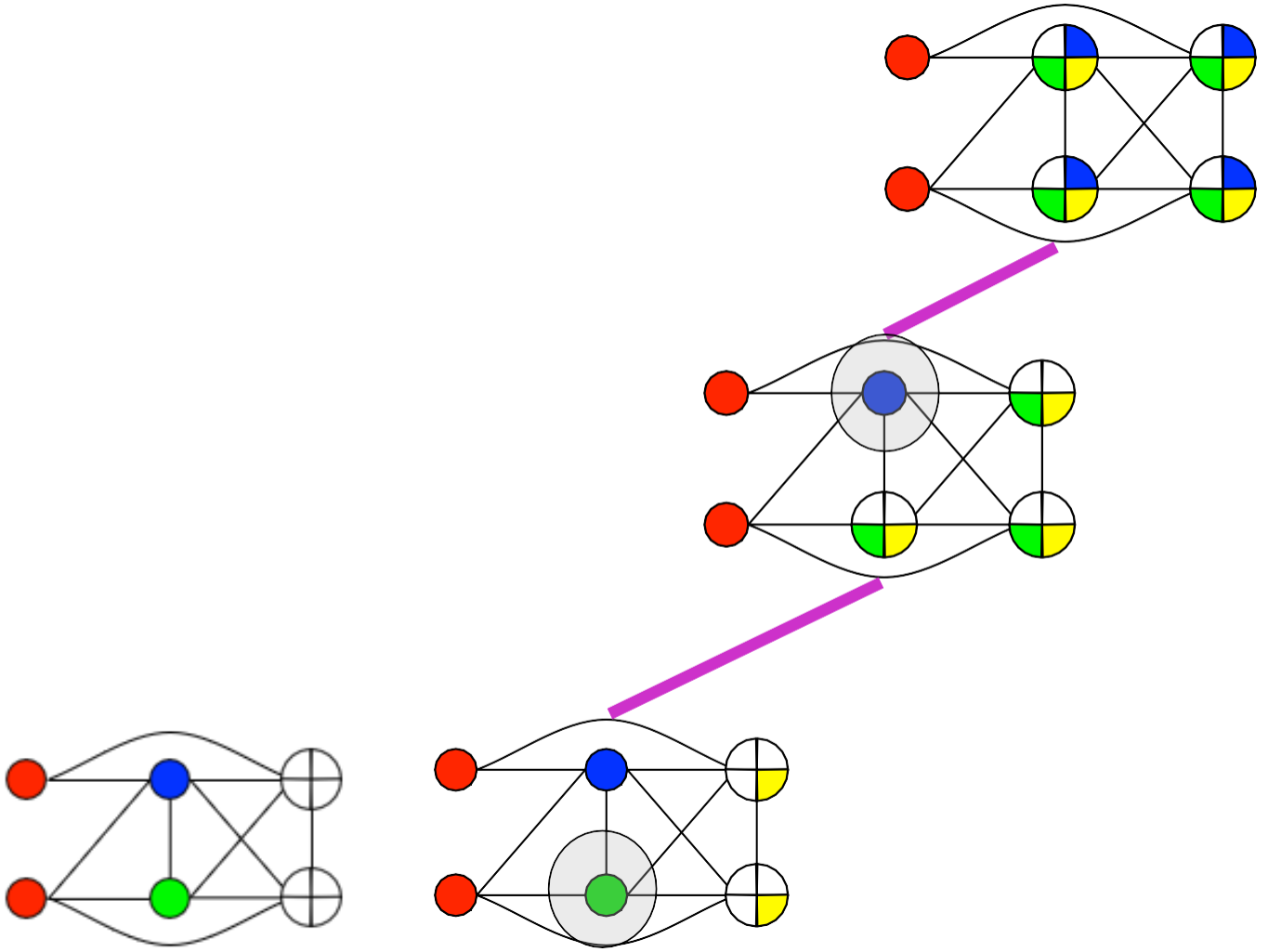
Graph Coloring CP



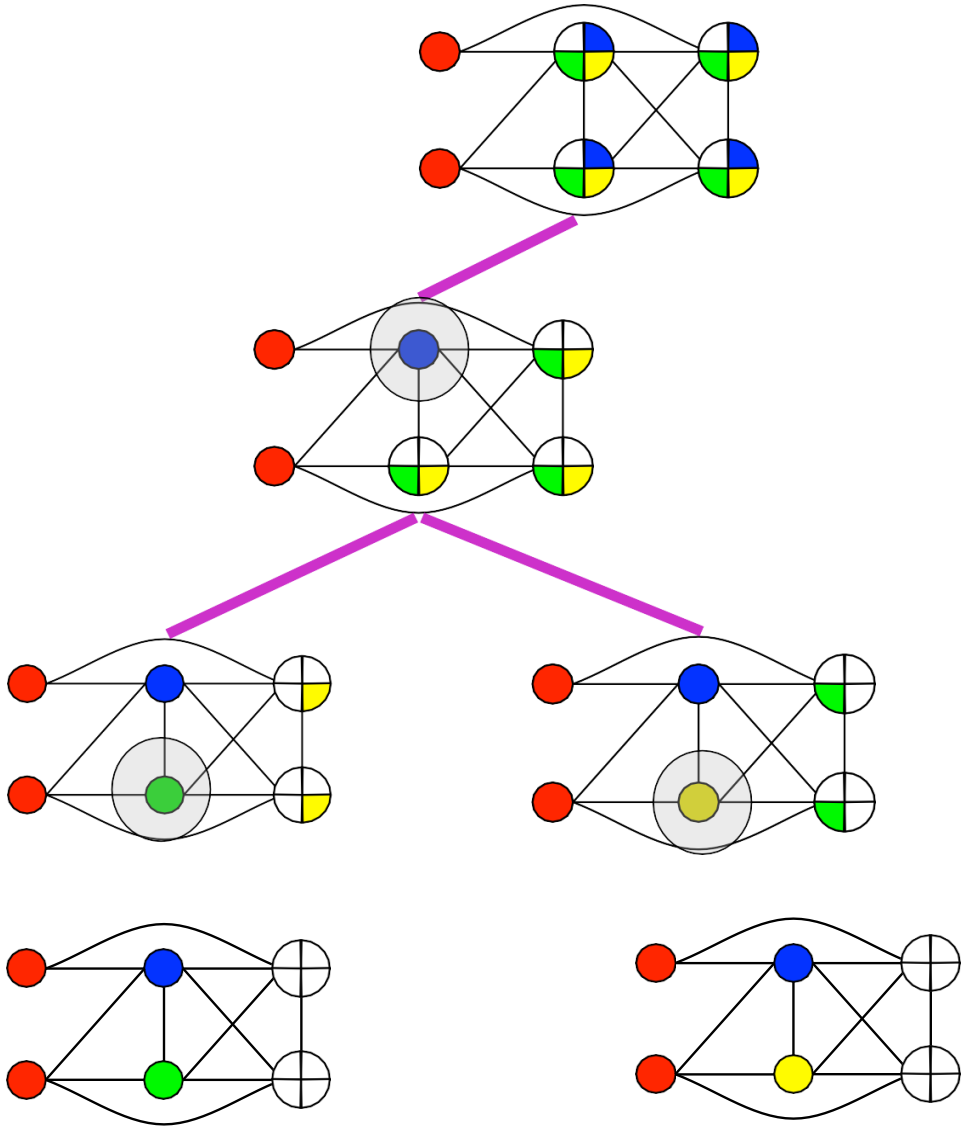
Graph Coloring CP



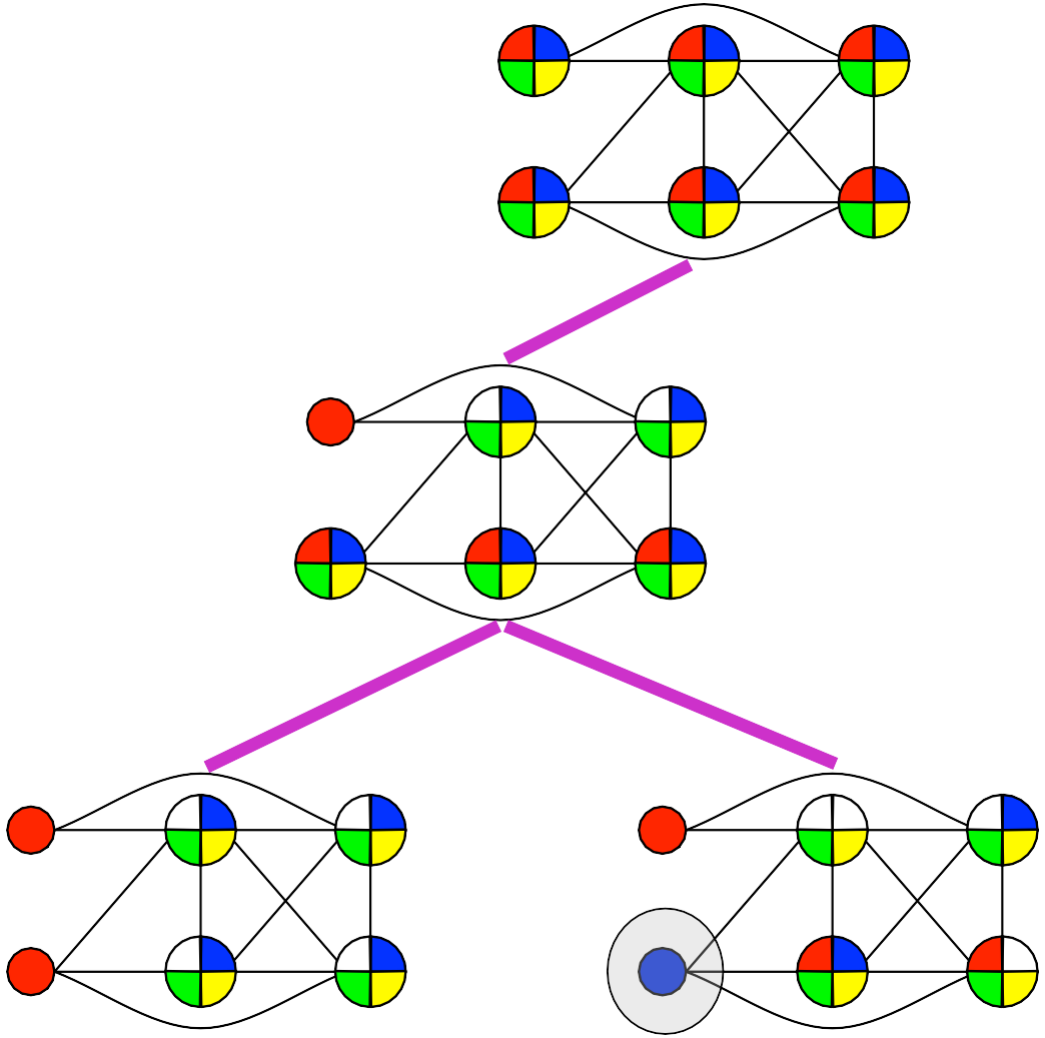
Graph Coloring CP



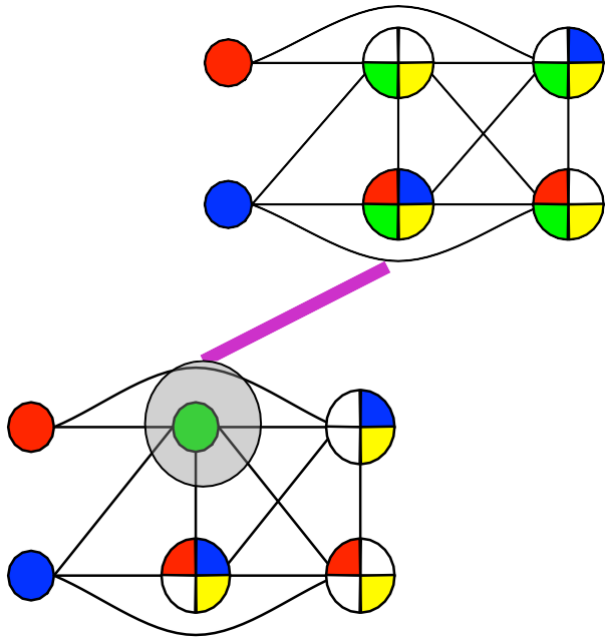
Graph Coloring CP



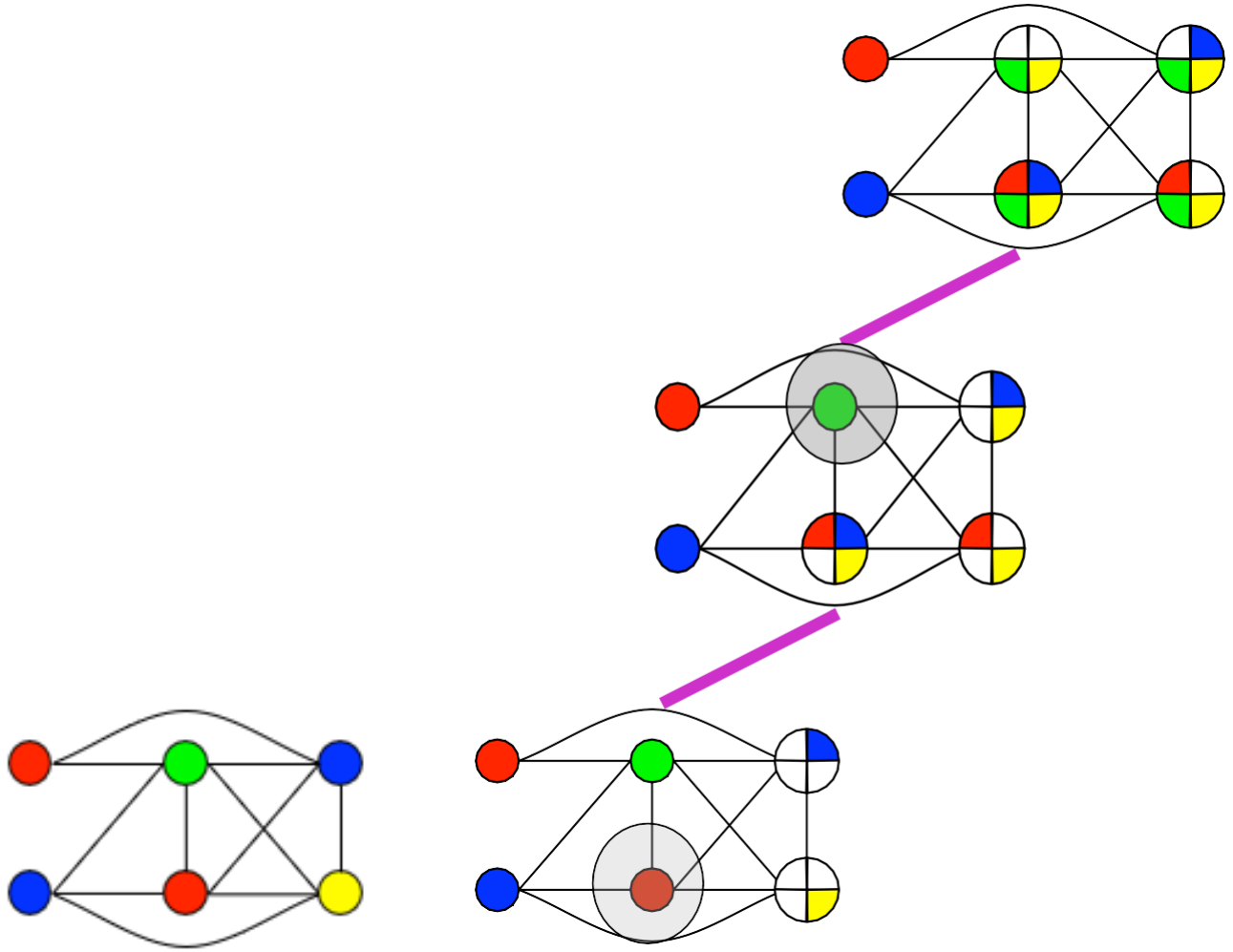
Graph Coloring CP



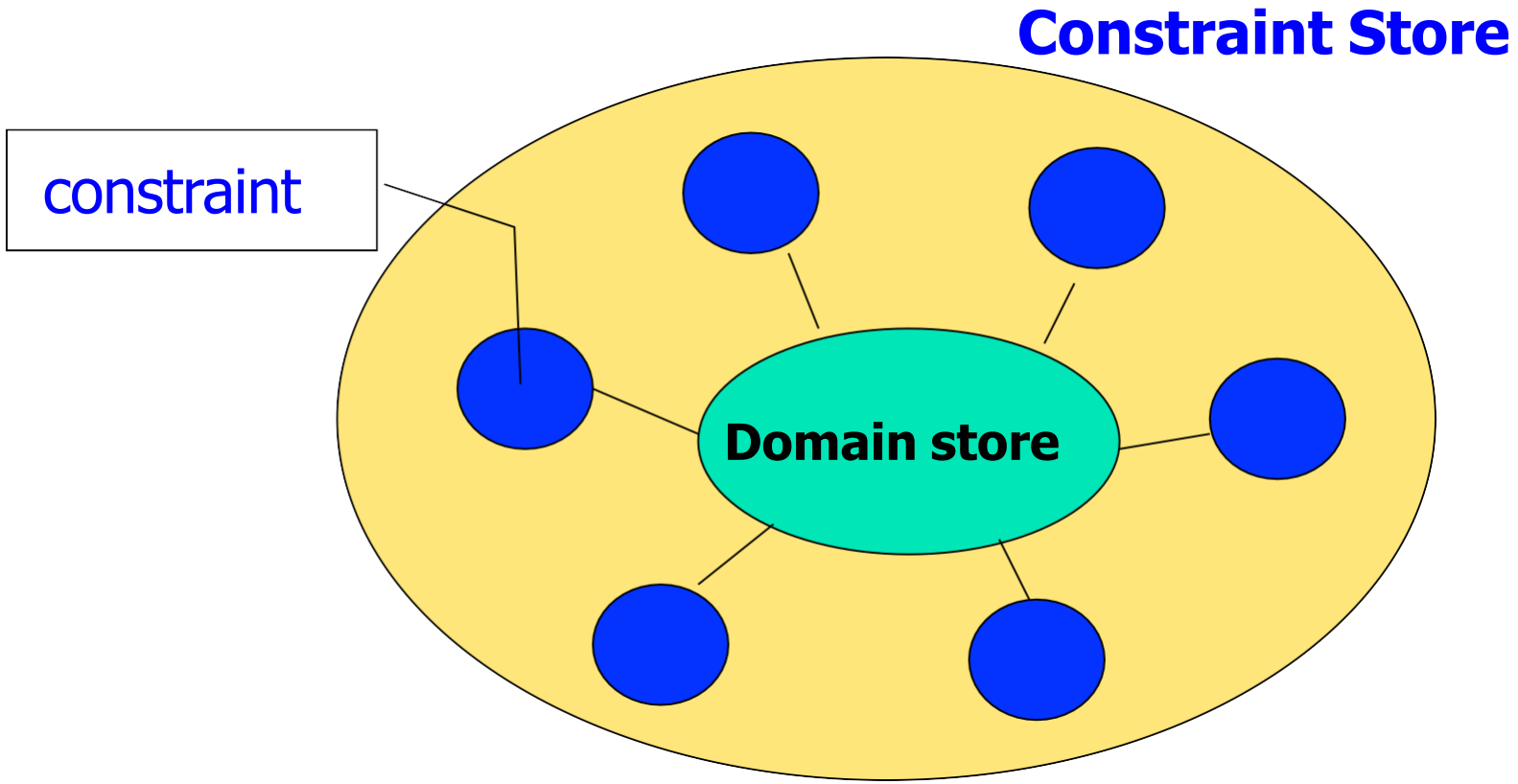
Graph Coloring CP



Graph Coloring CP



Computational Model



- Domain store
  - For each variable: what is the set of possible values?
  - If empty for any variable, then infeasible
  - If singleton for all variables, then solution
- Constraints
  - Capture interesting and well studied substructures
  - Need to
    - **Check Feasibility** : Determine if constraint is feasible wrt the domain store
    - **Prune** : remove "impossible" values from the domains

- General (fixpoint) algorithm is

```
repeat
   select a constraint c
   if c is infeasible wrt the domain store
      return infeasible
   else
      apply pruning algorithm of c
until no value can be removed
```

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - What is CP ?
  - Constraints and propagation
  - **Modeling**
  - Searching
  - Global constraint
  - Computation domain
- Examples with Or-Tools

Central theme

Solution process
=
Model + Search

- The model
  - what the solutions are and their quality
- The search
  - how to find the solutions

Example : SEND + MORE = MONEY

- Replace each letter by a different digit so that

```
  SEND
+ MORE
------
 MONEY
```

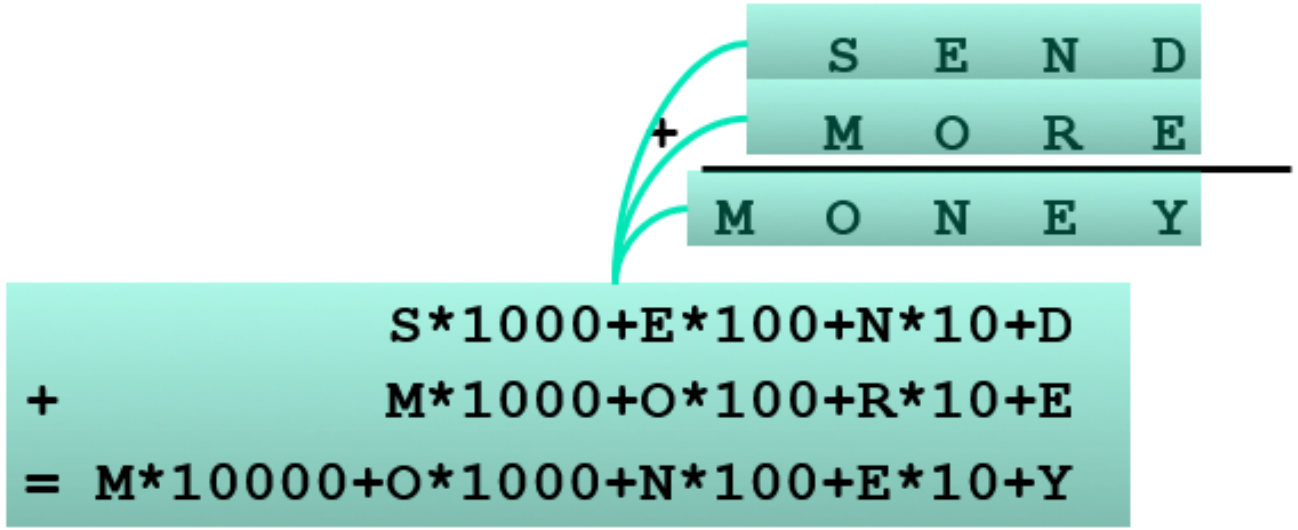
  is a correct sum.
  - Unique solution:

```
  9567
+ 1085
------
 10652
```

- Variables: S, E, N, D, M, O, R, Y
- Domains:
  - [1::9] for S, M,
  - [0::9] for E, N, D, O, R, Y .

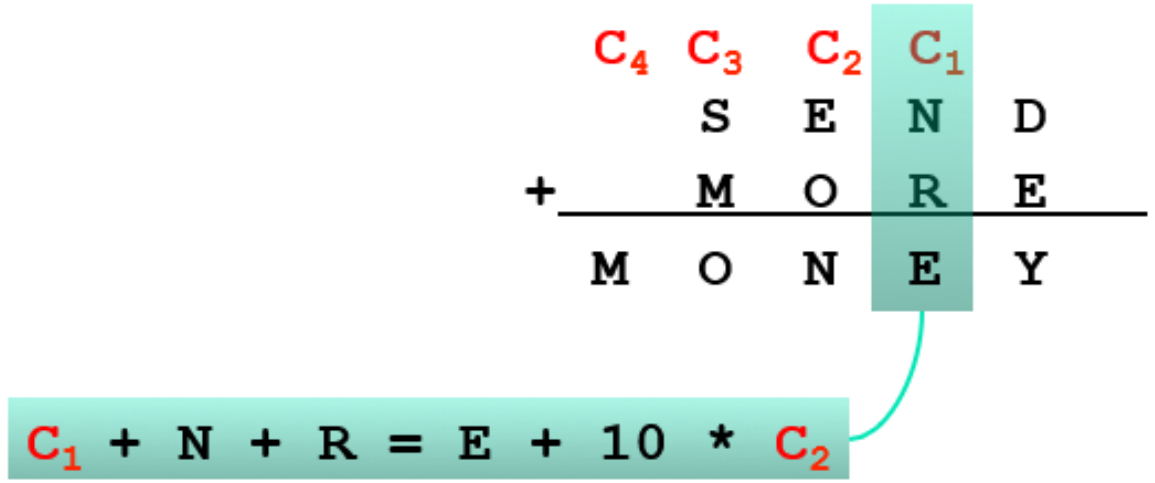
Example : SEND + MORE = MONEY

- 1 equality constraint
  - 1000 S + 100 E + 10 N + D +
    1000 M + 100 O + 10 R + E
    = 10000 M + 1000 O + 100 N + 10 E + Y



Example : SEND + MORE = MONEY

- **Or** 5 equality constraints.
  - Use carry variables C1, …, C4 ∈ [0..1]
  - D + E = 10 C1 + Y
  - C1 + N + R = 10 C2 + E
  - C2 + E + O = 10 C3 + N
  - C3 + S +M = 10 C4 + O
  - C4 = M



Example : SEND + MORE = MONEY

- 28 disequality constraints.
  - $x \neq y$ for $x, y \in \{S;E;N;D;M;O;R;Y\}$
- **Or** a single constraint for disequalities.
  - For variables x1,…, xn with domains D1,…,Dn : all_different(x1, …, xn) := {(d1,…,dn) | $d_i \neq d_j$ for $i \neq j$}
  - Use `all_different(S,E,N,D,M,O,R,Y)`
- **Or** modeling it as an IP problem.
  - For $x, y \in \{S;E;N;D;M;O;R;Y\}$ transform $x \neq y$ to
  - $x - y <= 10 - 11\ z_{xy}$
  - $y - x <= 11\ z_{xy} - 1$
- where $z_{xy} \in [0..1]$; $z_{xy}=1 \Rightarrow x<y$
  - Disadvantage: 28 new variables.

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - What is CP ?
  - Constraints and propagation
  - Modeling
  - **Searching**
  - Global constraint
  - Computation domain
- Examples with Or-Tools

Branching

- Once constraint propagation is done, apply the search method

Choose a variable x with non-singleton domain (d_1, d_2, … d_i)

For each **d** in (d_1, d_2, … d_i)

  try to solve the problem with the additional constraint **x=d**

Central theme

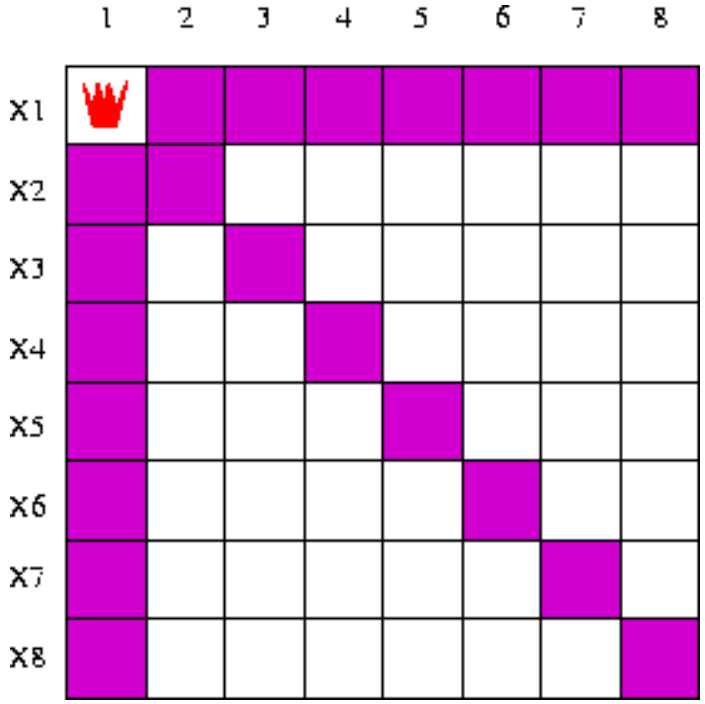
Solution process
=
Model + Search

- The search
  - nondeterministic program + exploration strategy
  - exploration strategy: DFS, LDS, BFS, …

The Queen Problem

Assuming domain consistency

Propagation of
queen[1] <> queen[2]
queen[1] <> queen[3]
…
queen[1] <> queen[8]

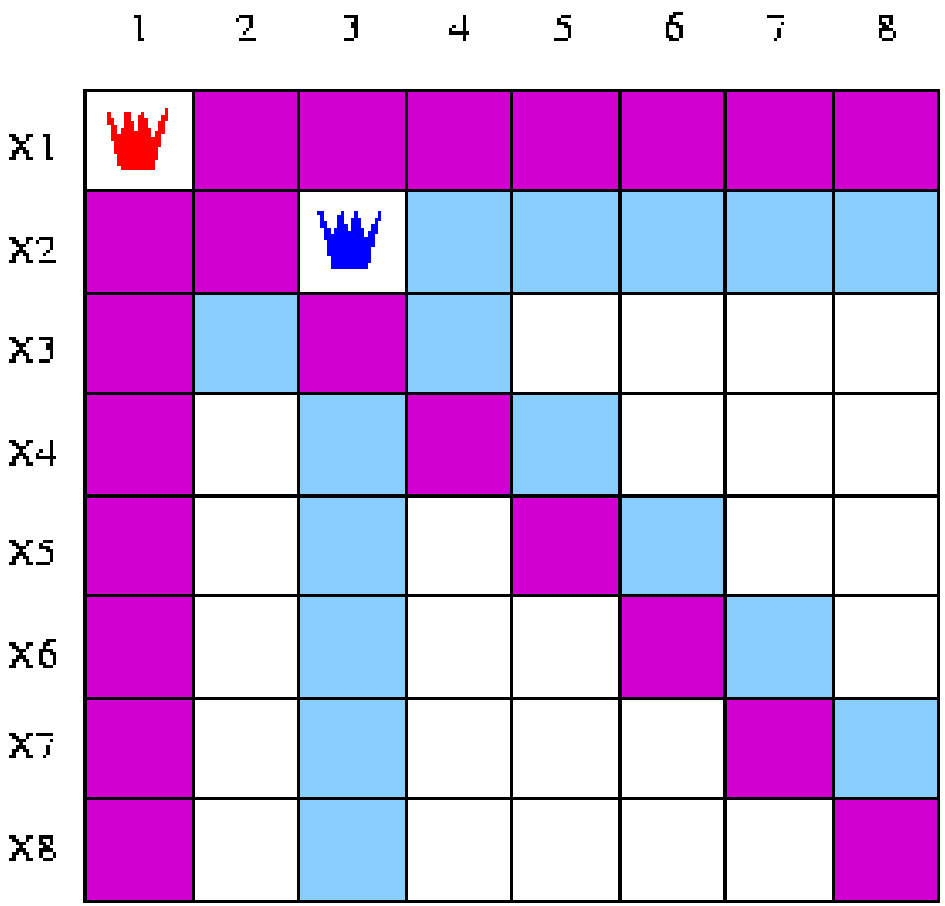


Consequence of
queen[1]=1

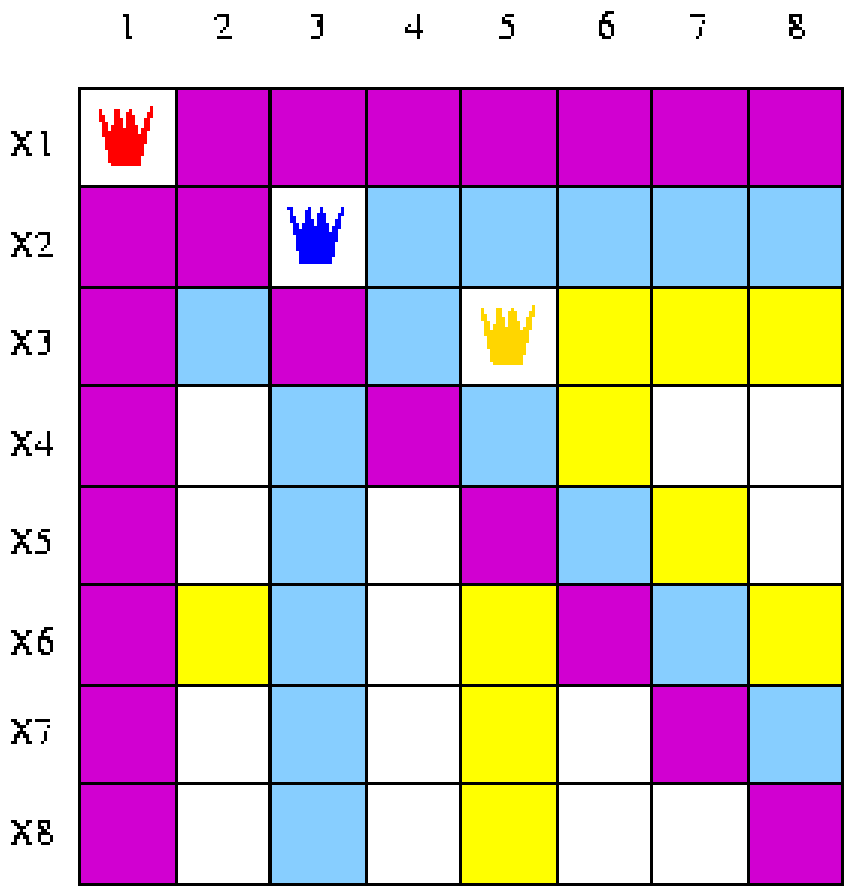
Propagation of
queen[1]+1<> queen[2]+2
queen[1]+1<> queen[3]+3
…
queen[1]+1 <> queen[8]+8

No more pruning
Place another queen
Questions
Which one ?
On which tile ?

The Queen Problem



The Queen Problem



The search

Search Procedure

=

Nondeterministic Program + Exploration Strategy

- Nondeterministic program
  - specify (implicitly) an **and-or tree**
- Exploration strategy
  - specify **how to explore the tree**

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - What is CP ?
  - Constraints and propagation
  - Modeling
  - Searching
  - **Global constraint**
  - Computation domain
- Examples with Or-Tools

Global Constraints

- Recognize some combinatorial substructures arising in many practical applications
  - alldifferent is a fundamental building block
  - many others (as we will see)
- Make modeling easier and more natural
  - Declarative
  - Compositionality

The alldifferent Constraint

- Most well-known global constraint.
  - `alldifferent(x,y,z)`
- states that x, y, and z take on different values.
  - x=2, y=1, z=3 would be ok,
  - but not x=1, y=3, z=1
- Why does it prune more than x≠y, y≠z, x≠z ?
- Very useful in many resource allocation, time tabling, sport scheduling problems

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - What is CP ?
  - Constraints and propagation
  - Modeling
  - Searching
  - Global constraint
  - **Computation domain**
- Examples with Or-Tools

Computation domains

- Propagators are associated to constraints and their underlying **computation domain**
- Basic computation domain : **finite domains**
- A set of constraints and their associated propagators (over a computation domain), is called a **constraint solver**
- Some possibilities to combine different computation domains (i.e. different solvers) within a CSP
- **Constraint programming engine** = constraint solvers + search

Computation domains

- Finite domains
- Linear real (rational) arthmetics
- Boolean
- Finite sets
- Terms
- Reals
- Intervals
- Graphs
- ...

Content

- A first challenge
- Constraint Satisfaction Problem
- **Constraint Programming**
  - What is CP ?
  - Constraints and propagation
  - Modeling
  - Searching
  - Global constraint
  - Computation domain
- **Examples with Or-Tools**

Example 1

- **Biến**
  - X = {X0, X1, X2, X3, X4}
- **Miền giá trị**
  - X0, X1, X2, X3, X4 ∈ {1,2,3,4,5}
- **Ràng buộc**
  - C1: X2 + 3 ≠ X1
  - C2: X3 ≤ X4
  - C3: X2 + X3 = X0 + 1
  - C4: X4 ≤ 3
  - C5: X1 + X4 = 7
  - C6: X2 = 1 => X4 ≠ 2

Solution

```python
from ortools.sat.python import cp_model

model = cp_model.CpModel()

x0 = model.NewIntVar(1, 5, 'x0')
x1 = model.NewIntVar(1, 5, 'x1')
x2 = model.NewIntVar(1, 5, 'x2')
x3 = model.NewIntVar(1, 5, 'x3')
x4 = model.NewIntVar(1, 5, 'x4')

model.Add(x2 + 3 != x1)
model.Add(x3 <= x4)
model.Add(x2 + x3 == x0 + 1)
model.Add(x4 <= 3)
model.Add(x1 + x4 == 7)

b = model.NewBoolVar('b')
model.Add(x2 == 1).OnlyEnforceIf(b)
model.Add(x2 != 1).OnlyEnforceIf(b.Not())
model.Add(x4 != 2).OnlyEnforceIf(b)

solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL:
    print('Maximum of objective function: %i' % solver.ObjectiveValue())
    print()
    print('x0 value: ', solver.Value(x0))
    print('x1 value: ', solver.Value(x1))
    print('x2 value: ', solver.Value(x2))
    print('x3 value: ', solver.Value(x3))
    print('x4 value: ', solver.Value(x4))
```

Example 2: CP + IS + FUN = TRUE

```
def CPIsFunSat():
    """Solve the CP+IS+FUN==TRUE cryptarithm."""
    # Constraint programming engine
    model = cp_model.CpModel()

    base = 10

    c = model.NewIntVar(1, base - 1, 'C')
    p = model.NewIntVar(0, base - 1, 'P')
    i = model.NewIntVar(1, base - 1, 'I')
    s = model.NewIntVar(0, base - 1, 'S')
    f = model.NewIntVar(1, base - 1, 'F')
    u = model.NewIntVar(0, base - 1, 'U')
    n = model.NewIntVar(0, base - 1, 'N')
    t = model.NewIntVar(1, base - 1, 'T')
    r = model.NewIntVar(0, base - 1, 'R')
    e = model.NewIntVar(0, base - 1, 'E')

    # We need to group variables in a list to use the constraint AllDifferent.
    letters = [c, p, i, s, f, u, n, t, r, e]
```

Example 2: CP + IS + FUN = TRUE

```
# Verify that we have enough digits.
assert base >= len(letters)

# Define constraints.
model.AddAllDifferent(letters)

# CP + IS + FUN = TRUE
model.Add(c * base + p + i * base + s + f * base * base + u * base +
          n == t * base * base * base + r * base * base + u * base + e)

### Solve model.
solver = cp_model.CpSolver()
solution_printer = VarArraySolutionPrinter(letters)

# Enumerate all solutions.
solver.parameters.enumerate_all_solutions = True
# Solve.
status = solver.Solve(model, solution_printer)

print()
print('Statistics')
print('  - status          : %s' % solver.StatusName(status))
print('  - conflicts       : %i' % solver.NumConflicts())
print('  - branches        : %i' % solver.NumBranches())
print('  - wall time       : %f s' % solver.WallTime())
print('  - solutions found : %i' % solution_printer.solution_count())
```

Example 2: CP + IS + FUN = TRUE

```
class VarArraySolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""

    def __init__(self, variables):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__variables = variables
        self.__solution_count = 0

    def on_solution_callback(self):
        self.__solution_count += 1
        for v in self.__variables:
            print('%s=%i' % (v, self.Value(v)), end=' ')
        print()

    def solution_count(self):
        return self.__solution_count
```

Example 3: N-Queen problem

```
def main(board_size):
  model = cp_model.CpModel()
  # Creates the variables.
  # The array index is the column, and the value is the row.
  queens = [model.NewIntVar(0, board_size - 1, 'x%i' % i)
            for i in range(board_size)]
  # Creates the constraints.
  # The following sets the constraint that all queens are in different rows.
  model.AddAllDifferent(queens)

  # Note: all queens must be in different columns because the indices
  #of queens are all different.
```

Example 3: N-Queen problem

```
# The following sets the constraint that no two queens can be on
# the same diagonal.
for i in range(board_size):
  # Note: is not used in the inner loop.
  diag1 = []
  diag2 = []
  for j in range(board_size):
    # Create variable array for queens(j) + j.
    q1 = model.NewIntVar(0, 2 * board_size, 'diag1_%i' % i)
    diag1.append(q1)
    model.Add(q1 == queens[j] + j)
    # Create variable array for queens(j) - j.
    q2 = model.NewIntVar(-board_size, board_size, 'diag2_%i' % i)
    diag2.append(q2)
    model.Add(q2 == queens[j] - j)
  model.AddAllDifferent(diag1)
  model.AddAllDifferent(diag2)
```

Example 3: N-Queen problem

```
  ### Solve model.
  solver = cp_model.CpSolver()
  solution_printer = SolutionPrinter(queens)
  status = solver.SearchForAllSolutions(model, solution_printer)
  print()
  print('Solutions found : %i' % solution_printer.SolutionCount())


class SolutionPrinter(cp_model.CpSolverSolutionCallback):
  """Print intermediate solutions."""

  def __init__(self, variables):
    cp_model.CpSolverSolutionCallback.__init__(self)
    self.__variables = variables
    self.__solution_count = 0

  def OnSolutionCallback(self):
    self.__solution_count += 1
    for v in self.__variables:
      print('%s = %i' % (v, self.Value(v)), end = ' ')
    print()

  def SolutionCount(self):
    return self.__solution_count
```

Example 4: Bài toán Phân bổ môn học

- Có N môn học 1, 2, …, N cần được phân bổ vào P học kỳ 1, 2, …, P
- Mỗi môn học i có số tín chỉ là $credit(i)$
- $L = \{(i, j)\}$: tập các cặp môn học (i, j) trong điều kiện tiên quyết (môn i phải được xếp và học kỳ trước học kỳ của môn j)
- Cho trước các hằng số α, β, λ, γ. Hãy tìm cách xếp N môn học vào P học kỳ sao cho
  - Tổng số môn học trong mỗi học kỳ phải lớn hơn hoặc bằng α và nhỏ hơn hoặc bằng β
  - Tổng số tín chỉ các môn học trong mỗi học kỳ phải lớn hơn hoặc bằng λ và nhỏ hơn hoặc bằng γ

Example 4: Bài toán Phân bổ môn học

Môn	1	2	3	4	5	6	7	8	9	10	11	12
Số tín chỉ	2	1	2	1	3	2	1	3	2	3	1	3

$3 \leq$ Số môn học mỗi học kỳ ≤ 3

$5 \leq$ Số tín chỉ các môn học mỗi học kỳ ≤ 7

Phương án phân bổ

Học kỳ	1	2	3	4
Danh sách môn	2, 5, 3	1, 6,10	4,7,8	9,11,12

2	1
6	9
5	6
5	8
4	11
6	12
2	7
3	10
5	7
8	11
4	12

Example 4: Bài toán Phân bổ môn học

- Biến
  - X[p,i] = 1: môn i được phân vào học kỳ p
  - D(X[p,i])= {0,1}
- Ràng buộc
  - X[q,i] = 1 → X[p,j] = 0, (i,j) ∈ L, 1 ≤ p ≤ q ≤ P
  - $\sum_{p=1}^{P} X[p,i] = 1$, với mọi i = 1,2,…,N
  - $\alpha \leq \sum_{i=1}^{N} X[p,i] \leq \beta$, với mọi p 1,2,.., P
  - $\lambda \leq \sum_{i=1}^{N} X[p,i]credit(i) \leq \gamma$, với mọi p = 1,2,.., P

Example 4: Bài toán Phân bổ môn học

```
N = 12
P = 4
credits = [2, 1, 2, 1, 3, 2, 1, 3, 2, 3, 1, 3]

orderCourseLst = [(0, 1), (1, 2)]

alpha = 3
beta = 3
lamb = 5
gamma = 7

model = cp_model.CpModel()

# Khai báo biến x[i,j] = 1 nếu môn học j được phân vào kỳ p
x = []
for i in range(P):
    t = []
    for j in range(N):
        t.append(model.NewIntVar(0, 1, 'x['+ str(i) + "," + str(j) + "]"))
    x.append(t)
```

Example 4: Bài toán Phân bổ môn học

```
#Ràng buộc: Mỗi môn học chỉ được phân vào duy nhất 1 kỳ
for j in range(N):
    model.Add(sum(x[i][j] for i in range(P)) == 1)

#Ràng buộc: Số lượng môn học trong một kỳ phải nằm trong [alpha, beta]
for i in range(P):
    model.Add(sum(x[i][j] for j in range(N)) >= alpha)
    model.Add(sum(x[i][j] for j in range(N)) <= beta)

#Ràng buộc: Số tín chỉ trong một kỳ phải nằm trong [lambda, gamma]
for i in range(P):
    model.Add(sum(x[i][j]*credits[j] for j in range(N)) >= lamb)
    model.Add(sum(x[i][j]*credits[j] for j in range(N)) <= gamma)
```

Example 4: Bài toán Phân bổ môn học

```
#Ràng buộc: Thứ tự các môn học
for item in orderCourseLst:
    i = item[0]
    j = item[1]

    for q in range(P):
        for p in range(q+1):

            b = model.NewBoolVar('b')
            model.Add(x[q][i] == 1).OnlyEnforceIf(b)
            model.Add(x[q][i] != 1).OnlyEnforceIf(b.Not())
            model.Add(x[p][j] == 0).OnlyEnforceIf(b)
```

Example 4: Bài toán Phân bổ môn học

```
### Solve model.
solver = cp_model.CpSolver()
status = solver.Solve(model)

if status == cp_model.OPTIMAL or status == cp_model.FEASIBLE:
    print(f'Total cost = {solver.ObjectiveValue()}')
    print()
    for i in range(P):
        for j in range(N):
            if solver.BooleanValue(x[i][j]):
                print(
                    f'Môn học {j} được phân cho kỳ {i}')
else:
    print('No solution found.')
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !