# Exercises

# X2.1 - Exercise: CSP **Modeling**

**Model** the following problems:

1) SEND + MORE = MONEY
2) 4-queens, generalize to N-queens
3) Magic squares N
4) Vending machine
5) Stable matching

For all of the above problems, you must model them as a CSP (or a COP if applicable)

# X2.2 - Exercise: vending machine

Problem statement:

Work out the change C to be given out by a vending machine, knowing that the user inserts the amount A (in Eurocent, or just c) to pay for a drink that costs D c.

Problem information:

- The machine takes and returns Euro coins, no less than 5c, i.e. 2€, 1€, 50c, 20c,10c and 5c.
- The machine is loaded with a number N of each type of coin
- C is an array indexed by the coin types (i.e. we have C[2E], C[1E]… C[5c])
- Extend the problem to a COP, i.e. minimise the number of coins

# X2.3 - Exercise: Stable Matching

5 men and 5 women with their preferences:

| Men's preferences | Women's preferences |
|---|---|
| 1: 1, 2, 4, 3, 5 | 1: 2, 3, 5, 4, 1 |
| 2: 3, 4, 1, 5, 2 | 2: 2, 4, 3, 1, 5 |
| 3: 4, 3, 5, 2, 1 | 3: 5, 3, 2, 4, 1 |
| 4: 1, 5, 2, 4, 3 | 4: 1, 5, 4, 3, 2 |
| 5: 5, 2, 3, 1, 4 | 5: 4, 3, 2, 1, 5 |

Lower position (number) indicates higher preference (i.e. it's a rank)

Find a ***stable*** marriage, i.e. a list of (m,w) pairs where no man-woman pair occurs in which **both** would rather marry each other than their assigned partner.

# (X2.3) - Stable Matching: modeling hints

- Represent **Wi** as the wife of man **i** and **Hj** as the husband of woman **j**. All **Wi** and **Hj** will range over **1..n**.
- Define a ranking which tells us the rank, for man **i**, of woman **j**. Conversely indicate the ranking for woman **i**, of man **j**. Do this for all applicable **i** and **j**.
- Define “integrity constraints” (mutual exclusion, reciprocity, monogamy)
- State the stability condition, both from the point of view of the men and of the women.
    - Hint: if man **m** prefers another woman **o** to his wife, then **o** must prefer her husband to **m** (and conversely for woman **w** and other man **o**)

# A first Choco Example

```
Solution: T=9, W=2, O=8, F=1, U=5, R=6,
```

```
      T W O          9 2 8
+     T W O     +    9 2 8
---------       ---------
= F O U R       = 1 8 5 6
```

Exercise:

- Find and display all 7 solutions

Helper: include the following in the Java code:

```
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solution;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.variables.IntVar;
```

# A variant

Repeat the previous exercise but with the puzzle:

```
    S E N D
+   M O R E
-----------
= M O N E Y
```

- How does it compare with hand-written code for this instance?

# Exercise: vending machine

Work out the change C to be given out by a vending machine, knowing that the user inserts the amount A (in €cent, or just c) to pay for a drink that costs B c. Problem information:

- The machine takes and returns Euro coins, no less than 5c, i.e. 2€, 1€, 50c, 20c, 10c and 5c.
- The machine is loaded with a number N of each type of coin
- C is an array indexed by the coin types (i.e. we have C[2€], C[1€]… C[5c])

Try with A=200 and B=135.

How many solutions? What is the "best"? How to compute it?

- (optional) extend the problem to a COP, i.e. minimise the number of coins

# Exercises

- Encode the N-queens problem.
  - Recall the possible models.
  - Which one do you choose?
  - How does this compare with your previous (pure Java) implementation?
- Encode the Magic Squares (N) problem
  - Can you solve N=3, 4, 5, 6 …
  - What is the runtime for each value of N?
  - What is the current limit under 5 minutes runtime?
  - How does this compare with your previous (pure Java) implementation?

# X4.1 - Exercises: cumulative global constraint

**Exercise A: are these instances valid solutions ?**

```
1) cumulative({ (1 2 3 3), (2 2 4 2), (4 1 5 1) }, 4)
2) cumulative({ (1 2 3 1), (4 1 5 2) }, 1)
3) cumulative({ (1 2 3 0), (1 2 3 4), (4 1 6 1) }, 4)
```

*cumulative({ (start, duration, end, height)... }, limit)*

**Exercice B: find all possible solutions**

```
cumulative({ ([1..9] [1] [1..8] [1])
             ([1..9] [2] [1..8] [2])
             ([1..9] [3] [1..8] [5])
             ([1..9] [4] [1..8] [7])}, 7)
```

Do this with a Java program using Choco and the `cumulative` constraint.

# X4.2 - Exercise: decompositions

Find all decompositions of an integer n >= 1. A decomposition of n is a tuple (a1,...,ak) s.t. ai >=1 and a1 + a2 + ... + ak = n. For n = 5:

(1 1 1 1 1) (1 1 1 2) (1 1 2 1) (1 1 3) (1 2 1 1) (1 2 2) (1 3 1) (1 4)
(2 1 1 1) (2 1 2) (2 2 1) (2 3)
(3 1 1) (3 2)
(4 1)
(5)

1) How many decompositions for 1? For 2, for 3? For a given n?
2) How would you solve this problem in Java without constraints?
3) Formalize as a CSP and write a Choco program to find all decompositions (respecting the above order if possible).

Hint: what about … a `regular` constraint to remove things unlike (1 2 2 0 0) or (5 0 0 0 0) or (3 2 0 0 0)? :)

# X4.3 - Problem: square box packing

Place a set of square boxes in a rectangular container.

- The container is sized N*M
- There are K boxes of side s1, s2, …, sk

Programming notes:

- Use the provided classes for the problem instances (SquarePackingInstance.java) and the abstract framework (SquarePackingAbstract.java) to guide the constraint problem setup
- Use the (SquarePackingSkeleton.java) file as a basis.

# X5.1 - Exercise

Use reified constraints to express a timetabling problem, where students must:

- Have 30 hours of classes in a week
- A weekday has 10 1-hour slots, 5 in the morning and 5 in the afternoon
- If there are more than 2 hours in the morning on one day, then there must be at most 2 hours in the afternoon, and conversely

# X5.2 - Exercise: Logistics (*)

In a weighted undirected graph, given an initial node (E) and a set of destination nodes (Di), find a subgraph that includes paths from E to all Di. Simplifying assumption: Di is a singleton set (i.e. only one element).

Identify the subgraph that has minimal cost (sum of the weights) while covering E and all Di.

Example data:

| | **Graph (N-N:C)** | **Start-{End}** | **Cost** |
|---|---|---|---|
| **1** | 1-2:4, 1-3:2, 2-3:5, 2-4:10, 3-5:3, 4-5:4, 4-6:11 | 1-{6} | 20 |
| **2** | 1-2:4, 1-3:2, 2-3:5, 2-4:10, 3-5:3, 4-5:4, 4-6:11 | 1-{~~5,~~6} | 20 |
| **3** | 1-2:6, 1-3:1, 1-4:5, 2-3:5, 2-5:3, 3-4:5, 3-5:6, 3-6:4, 4-6:2 | 1-{~~5,~~6} | 11 |

For case (3) the output should be {1,3,5,6}:11

# Exercise

- Encode the Sudoku of rank N problem.
  - Vary the search strategy

- Design a Sudoku generator
  - Provide means for assessing the difficulty of the problem
  - You may use the previous solver as a helper