CAB320 – *Constraint Satisfaction Problems*

Last modified on Friday, 15 April 2022 by f.maire@qut.edu.au

The first exercise requires you to model a practical problem as a *Constraint Satisfaction Problem* (CSP). The challenge is to identify what are the relevant variables and what are the constants of an instance of the problem. In a CSP, each state is represented as a partial assignment of values to variables (some variables don't get assigned a value immediately). For example, in a map coloring problem, we associate a variable to each region. These variables get assigned a color in such a way that two adjacent regions receive distinct colors.

The second exercise is a programming exercise where you are asked to complete a subclass of our generic search *Problem* class for contraint satisfaction problems. The third exercise presents a *local repair* approach to CSP. We start in a state with conflicting variables and iteratively try to reduce the number of conflicts.

Exercise 1 [non programming]

- Articulate a CSP formulation for the *Rectilinear floor planning problem*: find non-overlapping locations in a **large rectangle** for a set of **smaller rectangles**. Here we do not consider other constraints that would arise in practical applications. We are only interested in expressing with formulas the *non over-lapping* constraints the smaller rectangles have to satisfy. Assume that the smaller rectangles can be moved by translation only (no rotations) and that you are given the size of all rectangles.

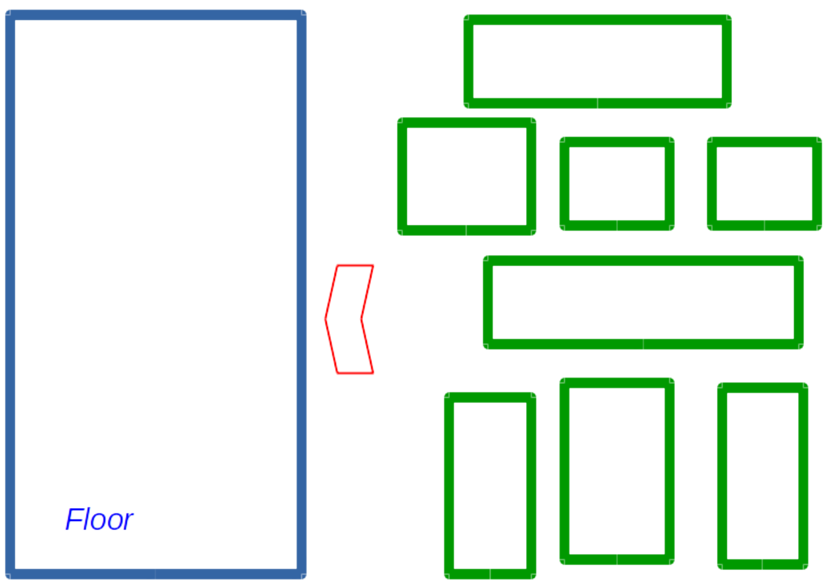

Illustration 1: Floor is in blue. Objective is to pack a set of non-overlapping green rectangles on the floor. We keep the rectangles axis parallel.

Exercise 2 (DFS for CSP)

- Open the file cab320_csp.py
- In the class CSP, how are state represented? What is the data structure used?
- What is an 'action' in the context of the CSP class? How is an action represented? (Hint: look at the method 'result')
- What are the 'neighbors' of a variable in 'csp_vars' ?
- How do you translate the 'goal_test' function in plain English?
- Complete the function 'actions' of the class CSP
- Test your CSP class implementation with a DFS on the MapColoringCSP problems.

Exercise 3 (Stochastic hill climbing)

In a search by stochastic hill climbing, we first start in a state where each variable has been assigned a value. This state might contains conflicting variables. After this initialization, we keep trying to reduce the number of conflicted variables. That is, we loop until we find a state with no conflict or we run out of time. In other words, the body of the main loop consists in the following steps;

- compute the list of conflicted variables (hint: use 'csp.conflicted_vards' method)
- if no variable are conflicted, we have reached a state that satisfy all the constraints
- Otherwise, we pick randomly one of the conflicted variables and assign to it a value that minimizes the number of conflicts (hint: use 'random.choice' and the function 'min_conflicts_value')

For the initialization step, we should also leverage the function 'min_conflicts_value' to start with a state that is better than a random assignment. We can create the initial state by iterating over the variables in 'csp.csp_vars' and assigning the value returned by 'min_conflicts_value(csp, var, current)' where 'current' is the current state and 'var' is the variable for which we want to find a value.

Implement this algorithm as the 'min_conflicts' function and test it on the MapColoringCSP problems.