1 Introduction

1.1 Constraint satisfaction problem

The constraint satisfaction problem is a general way in which decision-making problems can be represented and solved. It is defined as follows: **Given**

- A set of **decision variables** - A decision variable corresponds to a choice that must be made in solving a problem. For example in university timetabling, various things have to be decided, such as time for each lecture and venue for each lecture
- For each decision variable, a **domain** of potential values - Values in the domain of a decision variable correspond to the options for a particular choice. When a decision variable is assigned a single value from its domain, it is equivalent to the choice associated with that variable being made.
- A set of **constraints** on the decision variables. The **scope** of a constraint is a subset of the decision variables a constraint involves. Out of all possible combinations of assignments to the variables in its scope, a constraint specifies which assignments are allowed (satisfy the constraint) and which are disallowed (violet the constraint).

Find an **assignment** of values to variables such that all constraints are satisfied.

There are two main phases to solving problems with constraints: **Modelling** and **Solving**.

1.1.1 Constraint modelling

A constraint model maps the features of a given problem onto the features of a constraint satisfaction problem. In other words, the input problem must be modelled in a way that is represented with decisions variables, domains and constraints.

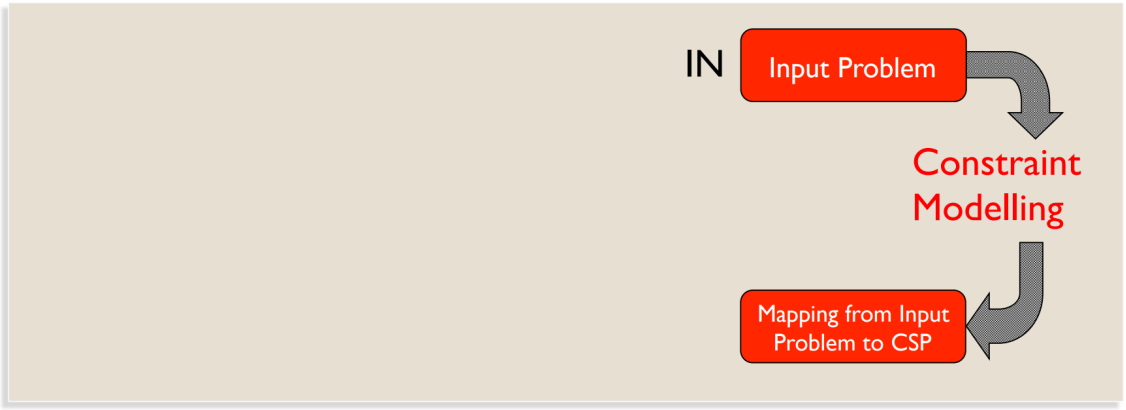

Figure 1: Constraint modelling is the first step in solving constraint problems.

1.1.2 Constraint solving

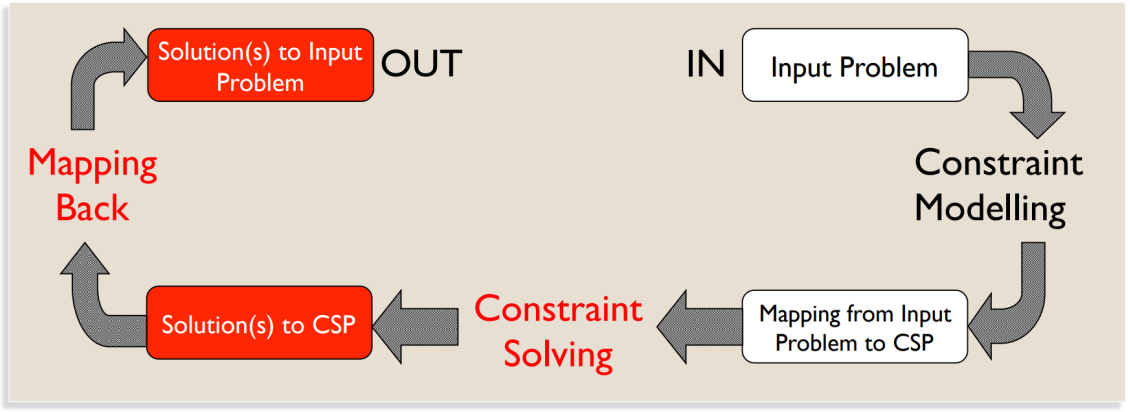

Figure 2: Full pipeline of constraint solving.

The modelled CSP is used as input to a constraint solver, which produces one or more solutions. To find solutions, a solver combines **search** and **deduction**.

1. A systematic search through the space of partial assignments makes guesses about variable assignments. The assignments are done incrementally to a subset of the variables and the solver must backtrack if the current assignments cannot lead to a solution.
2. Constraint propagation is the way to make deductions based on constraints and current domains. This is usually recorded as reductions in domains so that when a variable has a single value left in its domain, that is part of the solution.

1.2 Constraint optimisation problem

$$\text{COP} = \text{CSP} + \text{an objective function} \tag{1}$$

The constraint optimisation problem adds an **objective function** to a normal CSP. The goal is then to maximise or minimise the objective function. The goal is then to find a solution where all constraints are satisfied *and* the objective is optimised. Examples of these problems include minimising time taken in scheduling problems, or maximising profit. This gives four components to the constraint problem.

1. **Given** parameters
2. **Find** decision variables
3. **Such that** constraints
4. **Min/Maximising** objective

1.3 Problem classes

A problem class describes a family of problems, related by a common set of **parameters**. Examples of a problem class is the n-queens problem class, or sudoku problem class. The parameters are the same as each problem instance in the class has the same rules.

It is important to note that constraint solvers solver problem **instances**. An instance is specified by instantiating the parameters of a problem class to particular values. In the sample of sudoku, a sudoku problem instance is the instantiation of the filled in hints given on the cells. Further, an individual CSP/COP represents a problem instance.

1.4 Constraint representation

1.4.1 Table constraint

The table constraint is the most basic constraint available. It consists of listing the satisfying combination of assignments for all constraints and is known as the **extensional** representation. This is the most basic constraint as any constraint can be modelled in a table.

For example given two variables

$$X \text{ with domain } 1, 2, 3$$
$$Y \text{ with domain } 1, 2$$

and the constraint $X > Y$, the table constraint would look as follows:

Table(X, Y)
$\langle 2, 1 \rangle$
$\langle 3, 1 \rangle$
$\langle 3, 2 \rangle$

The issue with this basic constraint is that is can become both cumbersome and not practical. In an example of sudoku, where an AllDifferent constraint has to be imposed on 9 variable, the table would build to be 9! tuples large. The space to store the table tuples is not the only issue, as having to search through all the tuples would also take a significant amount of time. Because of this issue of combinatorial explosion, **intensional** representations were designed.

1.4.2 Intensional constraints

Rather than explicitly list all satisfying variable assignments, constraint solvers typically use intensional constraints to represent constraints like AllDifferent. This is done by the constraint solver by providing a library of commonly-occurring constraints that can be specified much more concisely.

However sometimes, the table (extensional) constraints must be used because it is the only sensible option, especially in examples where the constraint does not have an obvious algorithmic representation. This typically happens when the variables or domains are not numeric or easily represented as computing terms.

1.5 Constraint languages

Constraint programmers often do not want to work directly with CSP/COPs as they can be both large and too low-level to work with. Instead, the CSP/COPs are written as constraint programs (models) in constraint languages. The program/model is essentially a recipe to follow that reproduces a CSP/COP.

In a constraint language, decision variables and their domains must be declared. This is often done with arrays of variables, which allows for iteration to model problem classes. Further, constraint models written in constraint languages specify the problem *class* not the problem instance. The problem instances can be provided later as an instantiation of the parameters that the model specifies. Other common features in constraint languages include:

- Extensional constraints
- Equality, disequality, inequality

- Operators to build constraint expressions, for example +, -, AND, OR etc. These constraint expressions are then represented intensionally.

The model written in constraint languages can be parameterised to represent a problem class. This is done by giving values for the parameters to obtain an instance which corresponds directly to a CSP/COP. The CSP/COP can then be passed to a constraint solver to get the solution to that instance.

1.6 Crystal maze example

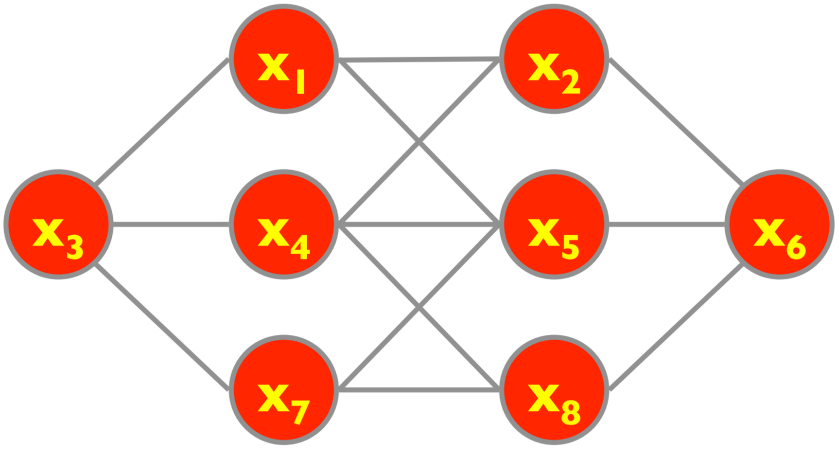

Figure 3: The crystal maze puzzle.

In the crystal maze puzzle, we must find $x_1 ... x_8$ with each domain $\{1, 2, 3, 4, 5, 6, 7, 8\}$ such that no adjacent nodes have adjacent values. For example, in the simple case of Constraint(x_1, x_2), the table constraint would look as follows:

Table(x_1, x_2)
$\langle 1, 1 \rangle$
$\langle 1, 3 \rangle$
$\langle 1, 4 \rangle$
...
$\langle 2, 2 \rangle$
$\langle 2, 4 \rangle$
...
$\langle 8, 8 \rangle$

Table 1: 50 possible combinations for each variable pairing that is connected.

The table initially is populated with 64 possible constraints for all values in the domain of x_1 and x_2. However, the constraint that says all connected variables cannot be assigned consecutive numbers removes some of the value assignments. The same constraint can be expressed in an intensional representation with the following equation:

$$|x_1 - x_2| \geq 1 \tag{2}$$

This works as it states that the values of x_1 and x_2 must have an absolute difference greater than 1. To be able to use the intensional representation, the constraint solver has to support the absolute value operator, subtraction operator and greater than constraint.