Handbook of Decision Making with Constraint Programming

January 27, 2024

Luca Polese
luca.polese@studio.unibo.it
Alma Mater Studiorum - Università di Bologna

Introduction

A combinatorial decision-making problem is a problem that is computationally difficult to solve (NP-Hard) that can only be solved by intelligent search. It's experimental, a time saver and it can reduce the cost of the environmental impact. The main purpose of CDM is that we want to make a decision within many combinations of possibilities. Each one is subject to restriction (that we will call constraints).

- we choose any solution that meet all constraints → constraint satisfaction
- we choose the best solution according to an objective → combinatorial optimization

Constraint Programming

It's a **declarative programming paradigm** for stating and solving combinatorial optimization problems.

User **models** a decision problem by formalizing:

- *unknowns* of the decision (decision variables : X_i);
- possible *values* for unknowns (domains : $D(X_i) = \{v_j\}$);
- *relations* between the unknowns (constraints : $r(X_i, X_i')$).

Pros of Constraint Programming

- It provides a rich language for expressing constraint and defining search procedures
- Easy to model
- Easy control of the search
- Main focus on constraints and feasibility
- Constraints reductions in the search space:
  - More constraints = more domain reductions → easier to solve.

Cons of Constraint Programming

- There is no focus on objective function
- There is no focus on optimality

Tradeoff between CP and ILP

The best choice to focus on optimality is to adopt hybrid models.

Solver

A **constraint solver** finds a *solution* to the *model* (or proves that no solution exists) by assigning a value to every variable ($X_i \leftarrow v_j$) via a search algorithm.

The constraint solver enumerates all value combinations (variable-value) via backtracking tree search and examines constraints.

Constraint Solver

- *Enumerates* all possible variable-value combinations via a systematic backtracking tree search.
- *Guesses* a value for each variable.
- During search, *examines* the constraints to *remove incompatible values* from the domains of the future (unexplored) variables, via **propagation**
- *Shrinks* the domains of the **future variables**