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**

Modeling

Formalization as a Constraint Satisfaction Problem

A CSP P is a triple $P =< X, D, C >$ where

- X is a set of decision variables $\{X_1, ..., X_n\}$,
- D is a set of domains $\{D_1, ..., D_n\}$ for X so that D_i is a set for X_i (non binary, with finite domain),
- C is a set of constraints $\{C_1, ..., C_n\}$ so that C_i is a relation over $X_j, ..., X_k$ denoted as $C(X_j, ..., X_k)$.

A solution to a CSP P is an n-tuple $A =< a_1, a_2, \ ... \ , a_n >$ where every a_i is an assignment of value to the variable X_i which satisfies all constraints simultaneously.
If the set of solutions is empty, the CSP is *unsatisfactory*.

Constraint Optimization Problems

CSP enhanced with an optimization criterion
Formally, $< X, D, C, f >$ where f is the formalization of the optimization criterion as an objective variable.
Our goal is to minimize/maximize f.

Constraints

We can use two different approaches to determine the constraints:

- Any constraint can be expressed by listing all allowed combinations.
  - General but inconvenient and inefficient with large domains.
- Usually it's better to use relations between objects.
  - Less general, but more compact and clear.

Properties of contraints

- The order of imposition does not matter
- Non-directional: a constraint between X and Y can be used on Y and X and vice versa
- Rarely independent, usually variables are shared between different constraints.

Types of constraint

Algebraic expressions	$x_1 > x_2$
Extensional constraints (*table* constraints)	$(X, Y, Z) \in \{(a, a, a), (b, b, b)\}$
Variables as subscripts (*element* constraints)	$Y = \text{cost}[X]$ (where cost is an array of parameters)
Logical relations	$(X < Y) \vee (Y < Z) \rightarrow C$
Global constraints	alldifferent$([X_1, X_2, X_3])$

Meta-constraints	$\sum_i (X_i > t_i) \leq 5$ (at most 5 variables should satisfy this constraint)

Modeling is critical

- Choice of variables and domains defines the search space size:
  $|D(X_1)| \times |D(X_2)| \times ... \times |D(X_n)| \rightarrow$ exponential!
- Choice of constraints defines:
  - how search space can be reduced
  - how search can be guided.

Symmetry in CSPs

> *Search can revisit equivalent states over and over again.*
>
> — Handbook of Constraint Programming

By **symmetry** we mean that:

> *Given a solution, which by definition satisfies all the constraints, we can find a **new solution** by applying any **symmetry** to the first solution we find. [dots] Why is symmetry important? The main reason is that we can **exploit symmetry** to **reduce** the amount of **search** needed to solve the problem.*
>
> — Handbook of Constraint Programming

A CSP can create many symmetrically equivalent search states each of which leads to a solution/failure and that will have many symmetrically equivalent states.

This is bad when we're proving optimality, infeasibility or looking for all solutions.

Permutation

Defined over a discrete set S as a 1-1 function π: $S \rightarrow S$, intuitively we re-arrange a set of elements.

Variable Symmetry

It's a permutation π of the variable indices s.t. for each (un)feasible (partial) assignment, we can re-arrange the variables according to π and obtain another (un)feasible (partial) assignment.

- Intuitively: permuting variable assignments.
- π identifies a specific symmetry.

Value Symmetry

It's a permutation π of the values s.t.for each (un)feasible (partial) assignment, we can re-arrange the values according to π and obtain another (un)feasible (partial) assignment.

- Intuitively: permuting values.
- π identifies a specific symmetry.

Symmetry Breaking Constraints

If we break the previous symmetries, then we reduce the set of solutions and search space.

- Not logically implied by the constraints of the problem.
- Attention: at least one solution from each set of symmetrically equivalent solutions must remain.

Dual Viewpoint

- Viewing a problem P from different perspectives may result in different models for P. Each model yields the same set of solutions. Each model exhibits in general a different representation of P. Can be hard to decide which is better.

Combined model

When we have multiple models that have relevant pros and cons we can opt for combining them. This way we can keep all the models and use channeling constraints to maintain consistency between the variables of the two models.

Benefits:

- Easier to express constraints
- Enhanced constraint propagation
- More options for search variables.