Constraint Programming

A gentle introduction

Dr.-Ing. Matthias Carlstedt

29 Sep 2025

Outline

What is Constraint Programming?

Constraint Programming (CP) is an optimization approach for discrete decisions.

- Many CP problems are **NP-hard**: worst-case runtime grows exponentially.
- CP succeeds where brute force/greedy fail due to *combinatorial explosion*.

Why brute force fails

- Tiny roster: 7 people, ≈ 4 options/day $\Rightarrow 4^7$ per day; over 7 days: $4^{49} \sim 10^{29}$.
- State space $\approx b^n$ (branching b, decisions n).

Perspective: hair $\sim 10^5$; sand $\sim 10^{19}$; stars $10^{22\text{–}24}$.

The CP model (compact)

$$(X, D, C, f)$$

- X: variables; D: finite domains; C: constraints; f: objective (optional).
- **CSP**: satisfy C. **COP**: satisfy C & optimize $f(x)$.

Domains: Boolean $\{0, 1\}$; integers; labels (e.g., Morning/Day/Night).
Arity: unary, binary, **global** (reusable structure).

Sudoku (CSP)

Variables

$$x_{r,c} \in \{1, \ldots, 9\} \quad \text{for rows } r = 1..9, \ \text{cols } c = 1..9.$$

Constraints

$$\begin{array}{rll} \text{Rows:} & \mathrm{AllDifferent}(x_{r,1}, \ldots, x_{r,9}) & \forall r \\ \text{Columns:} & \mathrm{AllDifferent}(x_{1,c}, \ldots, x_{9,c}) & \forall c \\ \text{Boxes:} & \mathrm{AllDifferent}\big(\{x_{r,c}\}_{r \in B_r,\ c \in B_c}\big) & \forall 3{\times}3 \text{ block} \\ \text{Clues:} & x_{r,c} = v & \text{where given (e.g. } x_{1,1} = 5) \end{array}$$

Objective

None (satisfaction problem).

Hospital roster (COP)

Variables

$$x_{e,d,s} \in \{0,1\} \quad \text{(employee } e \text{ works shift } s \text{ on day } d\text{)}, \qquad t_e \in \mathbb{Z}_{\geq 0}.$$

Constraints

$$\begin{aligned}
\text{Coverage:} \quad & \sum_e x_{e,d,s} = r_s & \forall d \in \{1,\dots,D\},\ \forall s \\
\text{Max one shift/day:} \quad & \sum_s x_{e,d,s} \leq 1 & \forall e,\ \forall d \\
\text{Weekly load (horizon):} \quad & t_e = \sum_{d=1}^{D} \sum_s x_{e,d,s}, \quad m \leq t_e \leq M & \forall e \\
\text{Rest rule:} \quad & x_{e,d,\mathrm{N}} + x_{e,d+1,\mathrm{M}} \leq 1 & \forall e,\ \forall d = 1,\dots,D-1
\end{aligned}$$

Objective

$$\min \ \left(\max_e t_e - \min_e t_e \right) \quad \text{(fairness gap)}.$$

Job shop scheduling (COP)

Variables

$$s_{j,k} \text{ start time of operation } k \text{ of job } j, \quad p_{j,k} \text{ its duration.}$$

(Equivalently: interval variables with `NoOverlap` per machine.)

Constraints

$$\begin{aligned} \text{Precedence:} \quad & s_{j,k+1} \geq s_{j,k} + p_{j,k} && \forall j, k \\ \text{Machine capacity:} \quad & \texttt{NoOverlap} \text{ of intervals on each machine } m \end{aligned}$$

Objective

$$\min \; C_{\max}, \quad C_{\max} = \max_{j,k} \left(s_{j,k} + p_{j,k} \right) \quad \text{(makespan).}$$

Demo

Run (terminal):

- `python 01_sudoku.py` (*AllDifferent* rows/cols/boxes)
- `python 02_roster.py` (*coverage, one/day, week load, fairness*)
- `python 03_jobshop.py` (*Intervals* + *NoOverlap, makespan*)

Solver: OR-Tools CP-SAT (integer CP) with time limits + optional parallel workers.

What to watch: feasibility first → KPIs (gap, loads, makespan) → search hints.

Global constraints — quick map (skip)

AllDifferent	Sudoku rows/cols/boxes (*strong propagation*)
NoOverlap / Intervals	One job per machine at a time (job shop)
Cumulative	Resource capacity K (parallel tasks)
Element	Index/select by a decision variable
Circuit / NoCycle	Tours, routing, acyclicity

Propagation, search, perturbation (skip)

- **Constraint propagation:** shrink domains early.
- **Search:** systematic backtracking with decision heuristics.
- **Perturbation:** Large Neighborhood Search (LNS), restarts, relax-and-fix.

Tools

- OR-Tools CP-SAT (Python) — Boolean/int vars, global constraints, fast.
- MiniZinc — high-level modeling, multiple backends (Chuffed, OR-Tools).
- Choco (Java), Z3/SMT (logic-heavy), CP Optimizer (IBM).

What CP is & a reusable recipe

What: variables X, domains D, constraints C, objective f (optional).

Recipe:

1. Decide variables and domains.
2. Encode rules (prefer global constraints).
3. Choose objective (makespan, fairness, cost).
4. Tighten (bounds, implied constraints, symmetry breaking).
5. Solve (propagation + search; restarts/LNS for larger cases).
6. Validate & iterate (hard rules, KPIs; refine/relax).

Further reading

- OR-Tools CP-SAT guide & Python examples: https://developers.google.com/optimization/cp
- OR-Tools Scheduling tutorials (roster, job shop): https://developers.google.com/optimization/scheduling
- OR-Tools GitHub examples (CP-SAT): https://github.com/google/or-tools/tree/stable/examples/python
- MiniZinc Tutorial (latest PDF): https://docs.minizinc.dev/en/latest/
- Global Constraint Catalog (definitions/propagation): https://sofdem.github.io/gccat/
- CSPLib problem library (benchmarks/models): http://www.csplib.org/ (e.g. 046: Meeting Scheduling)