BACHELOR'S PROJECT · UNIVERSITY OF PISA

SportSlot

A Resource Scheduling Optimization System for Elite Sports Centres

Constraint Programming with Google OR-Tools CP-SAT

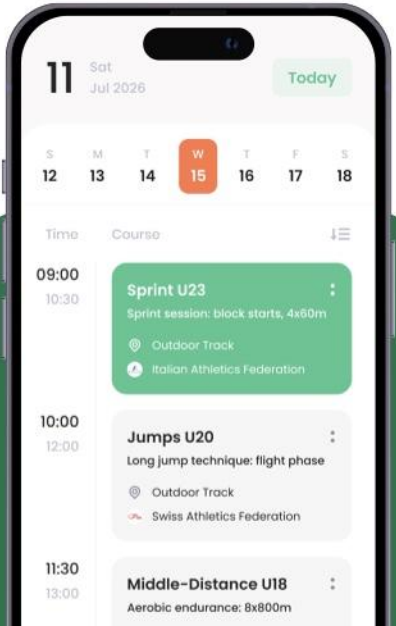
Lorenzo Galli

Dept. of Computer Science — University of Pisa

Internship at Kode S.r.l. — sportslot.it

Supervisor: Prof. Maria Grazia Scutellà

Company supervisor: Dr. Francesca Sbolgi



Presentation Outline

The Problem

Why sport-centre scheduling is a hard combinatorial problem.

The Model

A general mathematical formulation of resource allocation & scheduling, applied to a sports centre.

The System

From a Python/CP-SAT solver to a REST API and UI, containerized with Docker and deployed live.

The Experiments

System evaluation across different solver configurations.

Conclusions

Results, limitations, and future directions.

MOTIVATION

Manual scheduling breaks down as demand grows

Every week, a sport centre's scheduler faces dozens of requests from different organizations, each with its own specifications.

Fitting them all into a single timetable means checking every request against every resource, at every time slot, while respecting constraints and avoiding double-bookings.

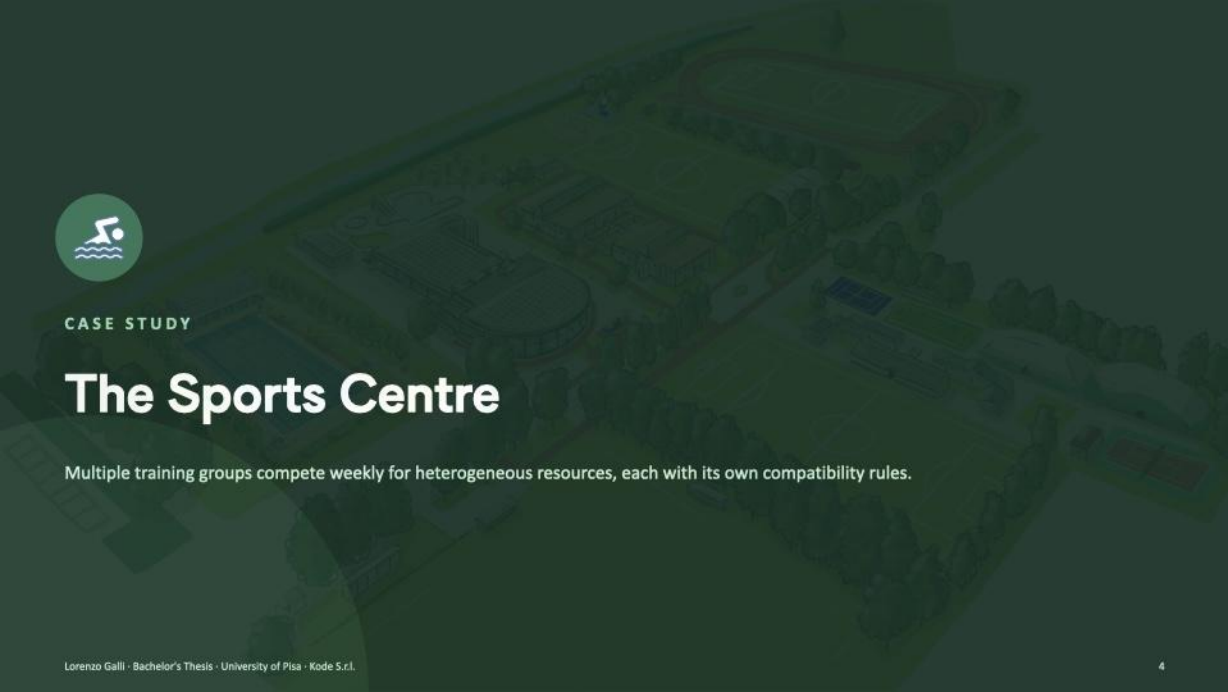
Doing this by hand is slow, error-prone, and rarely produces the best possible outcome, since a human planner has no systematic way to know if a better schedule even exists.

- **NP-hard:** No known algorithm can solve the general resource scheduling problem in polynomial time. In the worst case, the number of possible schedules grows exponentially.
- **Combinatorial:** Every extra request multiplies the number of feasible combinations.
- **Human impact:** A poor schedule wastes resources and athletes lose training time.

The mathematical approach

Constraint Programming turns these rules into a mathematical model, and lets a solver search the entire space of valid schedules automatically, returning feasible or optimal schedules.

"Optimality is guaranteed only when the solver proves no better solution exists."



CASE STUDY

The Sports Centre

Multiple training groups compete weekly for heterogeneous resources, each with its own compatibility rules.

Who asks, how, and for what?

Organizations & Groups

Each organization manages one or more groups. For example, a club's U18 team and U23 team train separately but belong to the same organization.

Requests & Resources

A group submits one or more requests. Each request specifies compatible resources, sessions per week, duration, and preferred time slots.

Time Windows

Each request specifies preferred days and hours, in 30-minute slots. Some are hard requirements, others are soft preferences the solver will try to honour.

The constraints

Every constraint is linear: no products between variables, just weighted sums.

Capacity

No resource r ever exceeds its capacity limit at any time slot.

Agent exclusivity

An agent can occupy at most one resource at any given time slot.

Availability

An assignment must fall within the resource's opening hours.

Session structure

Each request q requires exactly n separate sessions per week, each lasting m slots.

Rest time

At least n slots must elapse between consecutive sessions of the same request q.

Time windows & Compatibility

Some time windows are mandatory, while others are preferred and affect priority.

A general resource allocation model

*Defined as a **Binary Integer Linear Program (BILP)**, independent of the sports-centre domain.*

R — Resources

Available facilities, each with a fixed capacity and its own opening hours.

A — Agents

Requesters (e.g. training groups), each consuming *n* capacity units.

Q — Requests

Submitted by agents, each specifying its scheduling requirements.

T — Time slots

discrete slots across the weekly planning horizon (1 slot = 30 min).

Decision variable

$x_{q,r,t} = 1$ if request *q* is assigned to resource *r* at time slot *t*, 0 otherwise.

The objective function: priority-weighted coverage

The objective function is a linear weighted sum of the decision variables.

$$\text{maximize} \sum_{q,r,t} (w_q + \omega_{q,t}) \cdot x_{q,r,t}$$

*i.e., for each request **q**, scheduling one of its sessions earns w_q points, and doing so within a preferred time window earns a small additional bonus of $\omega_{q,t}$ points.*

The first sum rewards scheduling as many high-priority sessions as possible.

The second sum adds a small bonus when a session falls in a preferred time window, but this bonus never outweighs priority.

Three policies for handling contention — full coverage, partial coverage, or all-or-nothing — are compared later in the experiments.

Why Constraint Programming? Why CP-SAT?

Exact Methods vs. Heuristics

Exact methods systematically search the solution space and can prove a solution is optimal, but that search can become too slow on large instances. Heuristics are fast, but offer no such guarantee. CP-SAT, the solver used in this project, is an exact algorithm.

A Proven Guarantee

When CP-SAT returns OPTIMAL or INFEASIBLE, it is mathematically proven. Within a time limit, it may also return a FEASIBLE solution, not proven optimal. It uses heuristics only to decide where to search first and to reach an answer faster, never to guess a result.

Heuristics to Guide, Not to Decide

In CP-SAT, heuristics decide which variable to try next and which value to fix. Propagation fixes it and checks the consequences, continuing or backtracking on contradiction. On backtracking, the solver learns from the conflict and creates a new Boolean rule.

CP-SAT as the Right Choice

This is where *Constraint Programming* matters: CP-SAT natively supports high-level constraints (like *NoOverlap* and *Cumulative*) instead of requiring them to be encoded into linear constraints, as a plain MIP solver would.

Three ways of handling contention

Variant 1 — Full Coverage

Every session of every request must be scheduled.

Returns INFEASIBLE if any request can't be fully satisfied.

Variant 2 — Priority-Driven

Partial coverage allowed within requests.

Sessions are weighted by priority values, so higher-priority groups are scheduled first.

Variant 3 — All-or-Nothing

Each request is either fully scheduled or rejected entirely.

No partial sessions, organizations get a clear yes or no, not a half-filled schedule.

Two configurable parameters

num_workers

1 Deterministic and reproducible but slow on large instances

8 Balanced speed across cores

"max" or **null (default)** Uses all available CPU cores

More workers ⇒ non-deterministic results across runs.

search_branching heuristic

automatic CP-SAT's default: adjusts its variable-selection strategy on the fly based on search signals, without restarting.

fixed Branches on variables in the model's original order. It's predictable, but it can't adapt to what the search discovers.

portfolio Restarts periodically with a different heuristic each time, keeping all the information already learned.

lp Solves the LP relaxation at each node and branches on the variable closest to 0.5, the one it's least sure about.

The technology stack

Google OR-Tools

CP-SAT constraint solver

FastAPI

REST API backend

Streamlit

Interactive frontend

PostgreSQL

Relational database

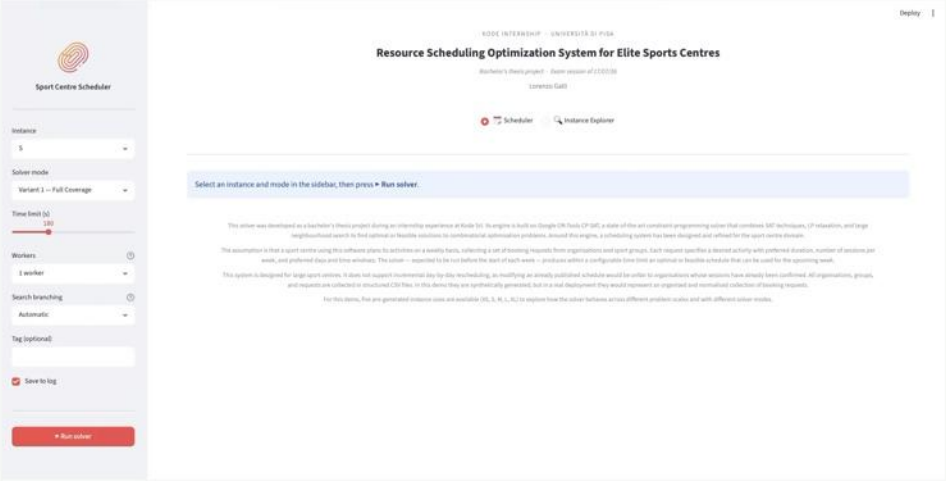
Docker Compose

3-container architecture

Caddy

HTTPS reverse proxy

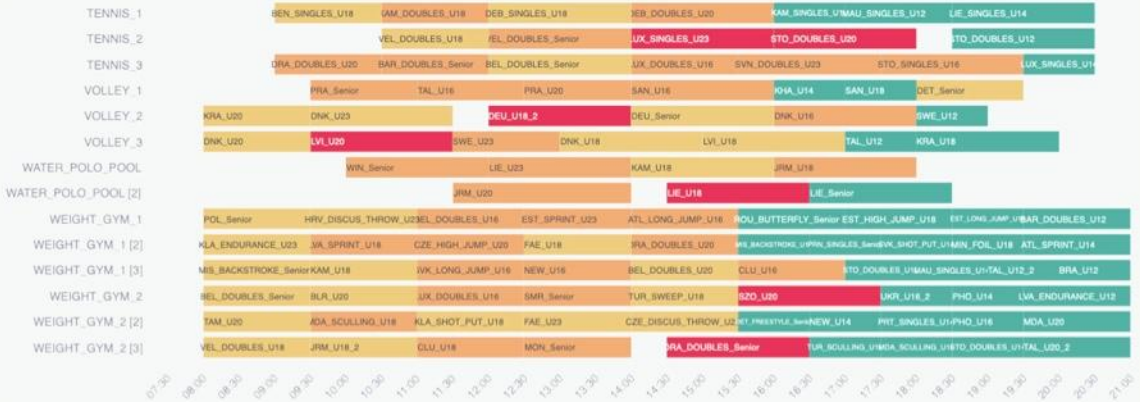
Live on sportslot.it



SCHEDULE EXAMPLE

High contention

Instance L, Variant 2, 10 workers



TESTING THE SOLVER

Experimental Evaluation

How solver variant, worker count, and branching strategy affect performance, tested across five synthetic instances (XS→XL).

DATA GENERATION

Data: 5 synthetic instances of increasing scale

No real booking data was available, so instances were generated with plausible, structured rules to enable controlled experimentation.

XS	S	M	L	XL
8 Orgs	16 Orgs	32 Orgs	64 Orgs	128 Orgs
21 Groups	51 Groups	90 Groups	168 Groups	350 Groups
37 Requests	84 Requests	149 Requests	276 Requests	562 Requests

Resource catalog (15–58 resources) stays fixed — the same physical centre under increasing demand.

EXPERIMENT 1

Solver variants under growing contention

Single worker, automatic branching, 180s time limit — coverage (%) across instances XS → XL.

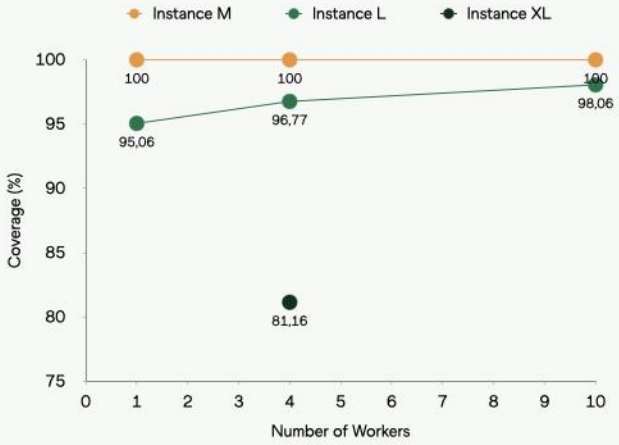
Instance	Variant	Status	Time (s)	Coverage (%)	Objective
XS	V1	OPTIMAL	1.30	100.00	14,346,750
XS	V2	OPTIMAL	1.17	100.00	14,346,750
XS	V3	OPTIMAL	1.16	100.00	14,346,750
S	V1	OPTIMAL	5.81	100.00	43,127,950
S	V2	OPTIMAL	18.95	100.00	43,127,950
S	V3	OPTIMAL	4.83	100.00	43,127,950
M	V1	OPTIMAL	51.99	100.00	75,769,400
M	V2	FEASIBLE	180.02	100.00	75,766,700
M	V3	OPTIMAL	119.75	100.00	75,769,400
L	V1	INFEASIBLE	14.22	N/A	N/A
L	V2	FEASIBLE	180.05	95.06	123,697,500
L	V3	FEASIBLE	180.04	90.76	122,413,200
XL	V1	INFEASIBLE	21.13	N/A	N/A
XL	V2	UNKNOWN	180.08	N/A	N/A
XL	V3	UNKNOWN	180.09	N/A	N/A

	Result
XS/S	Low contention, all variants reach OPTIMAL. V2 is noticeably slower.
M	Still 100% coverage, but V1/V3 reach OPTIMAL while V2 stalls at FEASIBLE (180s)
L	V1 turns INFEASIBLE (0%) as not all requests can be fully served. V2 (95.06%) beats V3 (90.76%) by spreading partial coverage.
XL	None of the three finds a feasible solution with one worker, motivating Experiment 2.

EXPERIMENT 2

Parallel workers: overhead, speed-up, accuracy or necessity?

Variant 2, automatic branching, 180s time limit — coverage (%) on M, L, XL with 1 / 4 / 10 workers.

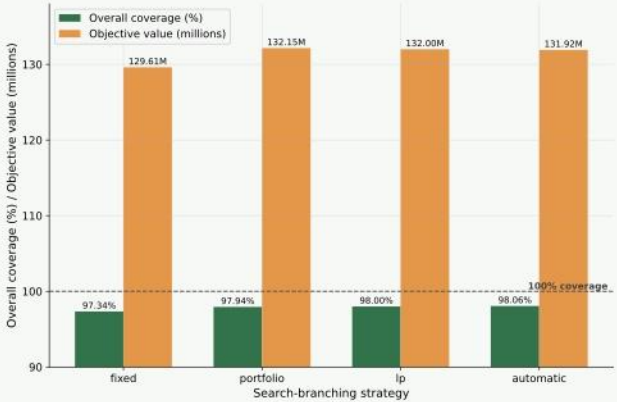


Instance	Workers	Status	Time (s)	Coverage (%)
M	1	FEASIBLE	180.02	100.00
M	4	FEASIBLE	180.18	100.00
M	10	OPTIMAL	103.13	100.00
L	1	FEASIBLE	180.05	95.06
L	4	FEASIBLE	180.27	96.77
L	10	FEASIBLE	180.93	98.06
XL	1	UNKNOWN	180.08	N/A
XL	4	FEASIBLE	181.94	81.16
XL	10	OUT OF MEMORY	N/A	N/A

EXPERIMENT 3

Branching strategy on a contended instance

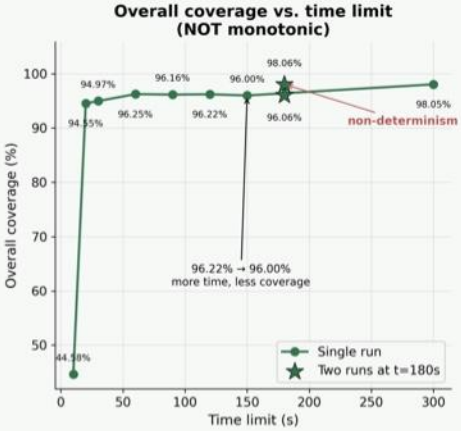
Instance L, Variant 2, 10 workers, 180s limit — comparing the four CP-SAT branching strategies.

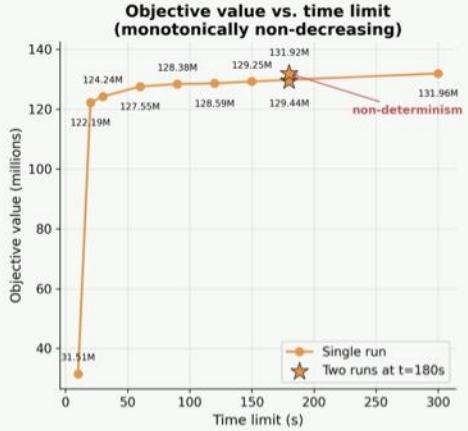


Branching	Sessions	Requests	Coverage (%)	Objective
Fixed	861/899	273/276	97.34	129,610,650
Portfolio	875/899	272/276	97.94	132,145,150
LP-based	876/899	272/276	98.00	131,995,850
Automatic	874/899	273/276	98.06	131,915,350

Time limit convergence

Instance L, Variant 2, 10 workers, automatic branching — coverage and objective across nine time limits (10s→300s).





DISCUSSION

What the experiments show

Relaxation matters only under contention

At low load, all three variants agree. Under higher contention, V2 handles it best, spreading partial coverage broadly.

Workers: from optimization to requirement

On small instances, multithreading is pure overhead. On XL, parallelism is the difference between finding a solution or not.

Branching is close, except fixed

Automatic, portfolio and LP-based search are near-equivalent. Fixed search consistently underperforms.

Recommended default

Variant 2, as many workers as memory allows, and automatic branching are the best balance of speed and accuracy.

WRAPPING UP

What we have build

1 A reusable BILP formulation

A reusable BILP formulation of resource allocation & scheduling, independent of the sports domain.

2 A structured Data Model & Dataset

Five synthetic instances (XS→XL) generated with plausible operational rules, replaceable with real data.

3 A working CP-SAT Engine

Three solver variants, exposed via REST API, Streamlit UI, and Docker deployment.

4 Experimental Evaluation

Solver variant, worker count, and branching strategy tested across five instances of increasing scale.

HONEST ASSESSMENT

Limitations

- Entirely synthetic data — no real sports-centre bookings were available to validate the generation rules against.
- Multi-worker runs are non-deterministic — only a limited slice of the configuration space was tested.
- Ten workers exceeded available memory even on a 16GB dev machine at XL scale.
- Single sports centre at a time — though it can be deployed as a separate Docker container per client.

LOOKING AHEAD

Future work

From Demo to Daily Use

Integrate the system into a real sports centre's operational workflow, as a decision-support tool.

Auto-tuned Configuration

Let the system pick workers & branching strategy automatically, based on available hardware.

Multi-Domain

Extend the domain-independent model to healthcare shift planning, school timetabling and more.

Thank You

Questions & Discussion

Lorenzo Galli
Bachelor's Project · University of Pisa · Kode S.r.l.