OptaPlanner

Prepared by Diana Dimova & Rares Sucalescu

Agenda

1 - Introduction

2 - Theory

3 - Kahoot

4 - Demo

General Information

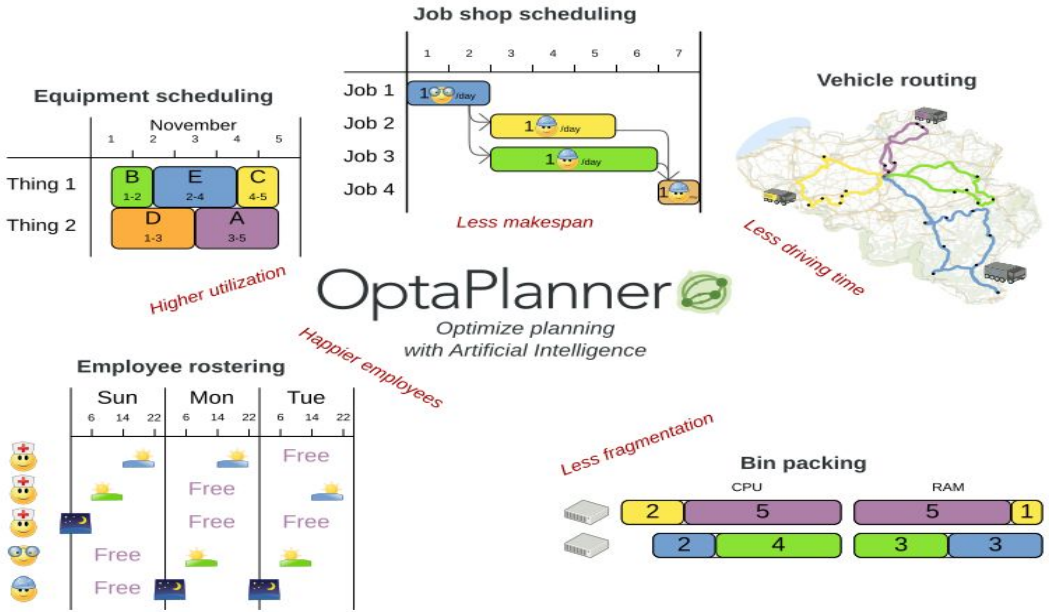
What is OptaPlanner

- An open-source software, written purely in Java.
- A lightweight, embeddable constraint satisfaction engine which optimizes planning problems.
- Runs on any JVM 8 or higher.
- OptaPlanner is available in the Maven Central Repository.

Compatibility

Use Cases

Main Use Cases



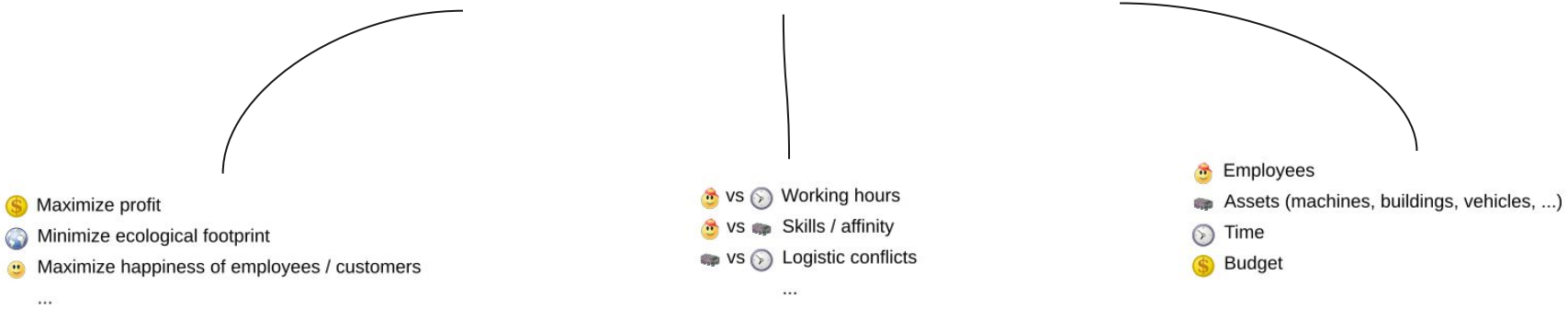
Additional Use Cases

- Employee shift rostering
- Agenda scheduling
- Educational timetabling
- Financial optimization
- Bin packing
- Job shop scheduling
- Cutting stock
- Sport scheduling

Planning Problem

What is the planning problem?

Optimize goals with **limited resources** under **constraints**



Planning Problem Constraints

Hard constraints:

e.g. 1 teacher cannot teach 2 different lessons at the same time.

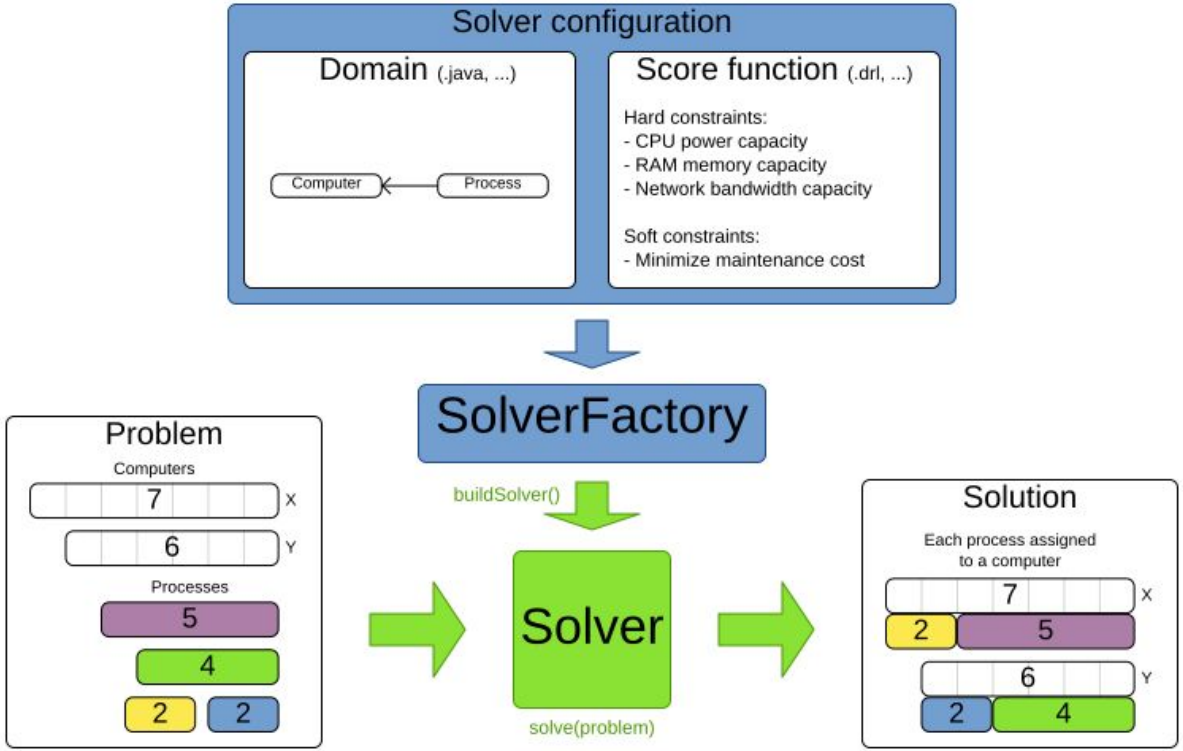
Soft constraints:

e.g. Teacher A does not like to teach on Friday afternoon.
e.g. Teacher B likes to teach on Monday morning.

Categories of solutions

- Possible solution
  - It is any solution, whether or not it breaks any number of constraints.
- Feasible solution
  - It is a solution that does not break any (negative) hard constraints.
  - Every feasible solution is a possible solution.
- Optimal solution
  - It is a solution with the highest score.
  - Planning problems tend to have 1 or a few optimal solutions. The optimal solution isn't feasible.
- The best solution
  - It is the solution with the highest score found by an implementation in a given amount of time.

How does it work?



Input/Output Overview

Domain Modelling

Essential Concepts

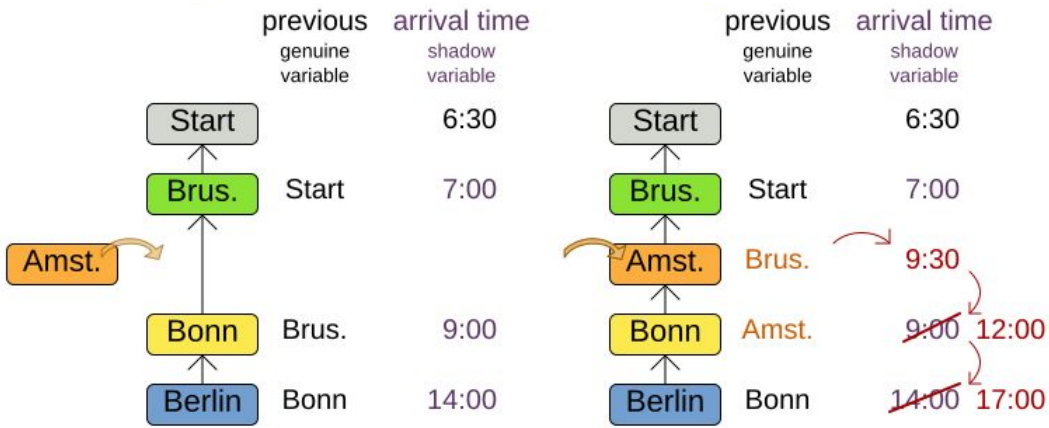
A **planning variable** is a JavaBean property (so a getter and setter) on a planning entity. It points to a planning value, which changes during planning.

A **shadow variable** is a planning variable whose correct value can be deduced from the state of the genuine planning variables.

A **planning entity** class is any class that has at least one shadow variable.

Planning Variable Listener

When a Customer's assignment changes,
the arrival time of that customer (and of its trailing customers) change too.



When a genuine planning variable changes,
then the Listener(s) change the shadow variable(s) accordingly.

Domain Modelling

BAD MODEL

ShiftAssignmet lacks of business identification. 2 planning variable make the search space a lot larger than necessary.

BAD MODEL

The number of employee shifts is impossible to determine in advance: it differs per solution. The nulls make the search space a bit larger than necessary.

GOOD MODEL

The number of employees per shifts is known in advance: it is part of the requirements.

What is the size of the search space?

Processes = 300

Computers = 100

$$|\text{Value set}|^{|\text{Variable set}|} = 100^{300}$$

How to make a good domain model?

- Make sure there are no duplications in your data model and that relationships between objects are clearly defined.
- Determine which relationships change during planning.
- If there is many-to-many relationship, replace it with a one-to-many and a many-to-one relationship to a new intermediate class.
- Make sure that the planning entity class has at least one problem property.
- Choose the model in which the number of planning entities is fixed during planning.

Solver

Solver

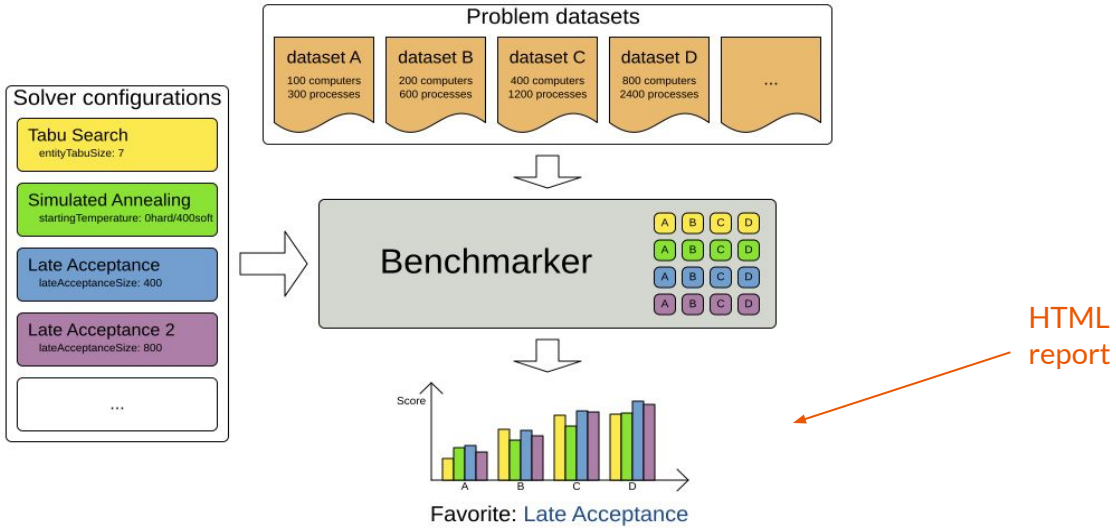
A **Solver** can use multiple optimization algorithms in sequence. Each optimization algorithm is represented by one solver **Phase**.

An optimization algorithm is great at finding new improving solutions for a planning problem, without necessarily brute-forcing every possibility. However, it needs to know the score of a solution and offers no support in calculating that score efficiently.

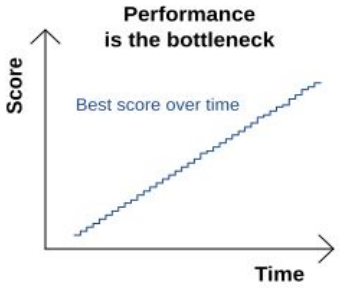
Benchmarking

Benchmarking

A Benchmarker tells what optimization algorithm is better to consider for the production.

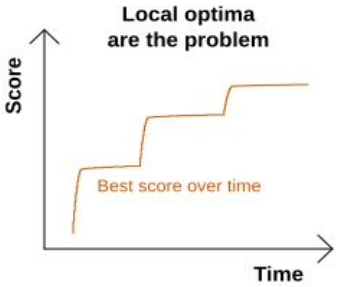


Benchmarking



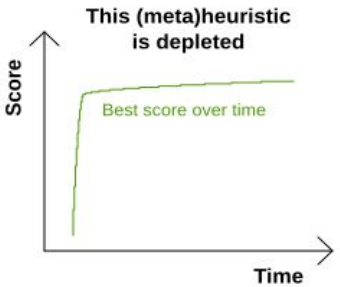
Observations:
Heavily improving every step,
No diminishing returns yet,
Solution not near optimal.

Recommendations:
Improve score calculation speed,
Use better hardware,
Give it more time



Observations:
Some moves are lucky because they break out of a local optima.

Recommendations:
Use constraint match statistics.



Observations:
Solution is likely near to optimal.

Recommendations:
Benchmark other algorithms.

Score Calculation

Score Calculation

- The Score is an objective way to compare two solutions.
- Higher Score => better solution
- The Solver aims to find the solution with the highest Score of all possible solutions.

Types of Score Calculation

Simple Java:

- Easy to write
- Slow

Incremental Java:

- Hard to write
- Hard to maintain
- Error prone
- Potentially fast

Types of Score Calculation p. 2

DRL:

- Learning curve
- Relatively easy to write
- Fast: implicit incremental calculation

Constraint Streams:

- Java Streams-like API
- Relatively easy to write (if familiar to Java Streams)
- Fast: implicit incremental calculation

QUESTIONS?

DEMO