Real-time constraint solving (with OptaPlanner)

by Geoffrey De Smet
OptaPlanner lead
A different kind of decisions?
Which processes fill up this computer as much as possible?

How did we find this solution?
First Fit by Decreasing Size

Processes CPU

A: 5
B: 3
C: 2
D: 1

Computer CPU

7

Not enough room

Not enough room

Optimal solution
First Fit Decreasing again...

Processes CPU

- **A** 5
- **B** 4
- **C** 3
- **D** 1

Computer CPU

- **7**

**Not enough room**

- 5

**Not optimal!**

**FAIL**

**Optimal solution**

- 3
- 4
This is... **NP Complete**

Can any algorithm find the optimal solution and scale out?
Find optimal solution and scale out for an NP-complete problem?

⇔ Is P = NP?

- Unresolved since 1971
- 1 000 000 $ reward since 2000
  - One of the 7 Millennium Problems
    (http://www.claymath.org/millennium-problems)
- Most believe P ≠ NP
  - ⇔ Impossible to find optimal solution and scale out
- 3000+ known NP-complete problems (wikipedia
NP-complete interconnection

Solve one use case
⇔ Solve all use cases
⇔ Prove $P = NP$
Use the right tool for the job.

- Insurance rate calculation: decision table
- License plate recognition: neural net
- Employee shift rostering: constraint solver

*Don't use a hammer on a screw.*
Constraint solver use cases...

- **Agenda scheduling**: doctor appointments, court hearings, maintenance jobs, TV advertisements, ...
- **Educational timetabling**: lectures, exams, conference presentations, ...
- **Task assignment**: affinity/skill matchmaking for tax audits, wage calc, ...
- **Employee shift rostering**: nurses, repairmen, help desk, firemen, ...
- **Vehicle routing**: route trucks, buses, trains, boats, airplanes, ...
- **Bin packing**: fill containers, trucks, ships, storage warehouses, cloud computers nodes, prisons, hospitals, ...
- **Job shop scheduling**: assembly lines for cars, furniture, books, ...
- **Cutting stock**: minimize waste while cutting paper, steel, carpet, ...
- **Sport scheduling**: football/baseball league, tennis court utilization, ...
- **Financial optimization**: investment portfolio balance, risk spreading, ...
Employee shift rostering
Employee rostering

Assign shifts to employees more efficiently.

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
<td>6</td>
<td>14</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>14</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Employees**
  - 1 shift per day
  - Forward rotation (enough time to sleep)
  - ≥ 10 hours
  - ≤ 5 consecutive shifts
  - Requires nurse skill
  - Requires engineering skill
  - ≥ 48 hours rest
  - Day off request
  - Shifts
  - Free
  - Free
  - Free
  - Free
  - No weekend work

**Users**
- Hospitals
- Courts of Justice
- Call centers
- Police and fire department

**NurseRostering benchmark**

Employee well-being **+53%**

OptaPlanner versus traditional algorithm with domain knowledge

Average

<table>
<thead>
<tr>
<th>Min/Max</th>
<th># datasets</th>
<th>Biggest dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>+19%</td>
<td>26</td>
<td>752 assignments 50 employees</td>
</tr>
<tr>
<td>+85%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Don't believe us? Run our open benchmarks yourself: https://www.optaplanner.org/code/benchmarks.html
What is constraint solving?
Implementation

1. Define domain
2. Define constraints
3. Solve
Input/Output overview employee rostering

Use 1 SolverFactory per application and 1 Solver per dataset.

Solver configuration

Domain (java, ...)
- Employee → Shift

Score function (.drl, ...)
- Hard constraints:
  - At most 1 shift/day
  - Skill requirements
- Soft constraints:
  - At least 10 hours between shifts
  - Day off requests

Problem

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
</tbody>
</table>

shfits
Day off request
Day off request

SolverFactory

buildSolver()

Solver

solve(problem)

Solution

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
</tbody>
</table>

Employee
Shift

employee
rostering
Define domain
Score Comparison Employee Rostering

Hard constraints always outweigh soft constraints.

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
</tbody>
</table>

- **Mon**
  - 6: Day off request
  - 14: Skill missing
  - 22: > 1 shift/day

- **Tue**
  - 6: Day off request
  - 14: Skill missing
  - 22: < 10h

- **Wed**
  - 6: Day off request
  - 14: Skill missing
  - 22: < 10h

- Highest score: 0 hard / -1 soft
Score calculation

- Easy Java (slow)
- Incremental Java (painful)
- Drools DRL (also incremental)
Architecture overview

The Solver wades through the search space of solutions efficiently. The ScoreDirector calculates the score of every solution under evaluation.

**Solver**
- Construction Heuristics
  - First Fit
  - First Fit Decreasing
  - Cheapest Insertion
  - ...
- Metaheuristics
  - Tabu Search
  - Simulated Annealing
  - Late Acceptance
  - Genetic Algorithms
  - ...

**ScoreDirector**
- Drools (rule engine)
- DRL
- Decision Table
- KIE Workbench
- Java
  - EasyScoreCalculator
  - IncrementalScoreCalculator

Find a better solution

Calculate the score of a solution

(Many times per ms)
Required skill constraint (easy Java)

```java
public class MyScoreCalculator
    implements EasyScoreCalculator<Roster> {

    public Score calculateScore(Roster roster) {
        int hardScore = 0;
        for (Shift shift : roster.getShiftList()) {
            Skill requiredSkill = shift.getSpot().getRequiredSkill();
            if (shift.getEmployee() != null // Employee lacks required skill
                && !shift.getEmployee().hasSkill(requiredSkill)) {
                // Lower hard score
                hardScore--;
            }
        }
        ...
        return HardSoftScore.valueOf(hardScore, softScore);
    }
```
Incremental score calculation
Calculating delta's is much faster than calculating the entire's solution's score.

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
</tbody>
</table>

BigO for n shifts

<table>
<thead>
<tr>
<th>Constraint</th>
<th>From scratch</th>
<th>Incremental</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required skill</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>At most 1 shift/day</td>
<td>$O(n^2)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

Check every shift:
$0 + 0 + 0 + 0 - 1 - 1 + 0 + 0$
Required skill score: $-2\text{hard}$

Calculation from scratch (easy java)

Check every shift again:
$0 + 0 + 0 + 0 - 1 + 0 + 0 + 0$
Required skill score: $-1\text{hard}$

Incremental calculation (inc. java, drools)

Check one shift (old & new)
$-2 + 1 - 0$
Required skill score: $-1\text{hard}$
Required skill constraint (Drools DRL)

```drl
rule "Required skill"
  when
    Shift(
      getEmployee() != null,
      // Employee lacks required skill
      !getEmployee().hasSkill(getSpot().getRequiredSkill()))
  then
    // Lower hard score
    scoreHolder.addHardConstraintMatch(kcontext, -1);
end
```
Input/Output overview employee rostering

Use 1 SolverFactory per application and 1 Solver per dataset.

Solver configuration

- **Domain** (java, ...)
- **Score function** (.drl, ...)

- Hard constraints:
  - At most 1 shift/day
  - Skill requirements

- Soft constraints:
  - At least 10 hours between shifts
  - Day off requests

---

Problem

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>22</td>
</tr>
</tbody>
</table>

SolverFactory

buildSolver()

Solver

solve(problem)

Solution

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When do we solve?

- Publish schedule weeks in advance
  - Affects family/social lives
- Ad hoc changes
  - Sick employees
  - Shift changes
Continuous planning

Replan at the start of every period. Plan 3 periods, but only share the first period.

<table>
<thead>
<tr>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
</tr>
</thead>
<tbody>
<tr>
<td>🏥</td>
<td>🎈</td>
<td>🎈</td>
<td>🎈</td>
<td>🏥</td>
<td>🎈</td>
<td>🎈</td>
<td>🏥</td>
<td>🎈</td>
<td>🎈</td>
<td>🎈</td>
<td>🎈</td>
<td>🎈</td>
<td>🎈</td>
<td>🏥</td>
<td>🎈</td>
</tr>
<tr>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
</tr>
<tr>
<td>tentative</td>
<td>draft</td>
<td>future</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

now

<table>
<thead>
<tr>
<th>now</th>
<th>history</th>
<th>tentative</th>
<th>draft</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
</tr>
<tr>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
</tr>
<tr>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
</tr>
<tr>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
<td>🌅</td>
</tr>
</tbody>
</table>
Vehicle routing

Assign the delivery order of vehicles more efficiently.

Driver wage 20$ / hour

Depot

Delivery locations

Capacity ≤ 20 ton

Optional
Can wait till tomorrow

Time window
Deliver between 8 AM and 10 AM

Armored vehicle

Expensive delivery

Users

Supermarkets & retail stores

Freight transportation

Buses, taxi's & airlines

Technicians on the road

VehicleRouting benchmark (Belgium datasets)

Driving time -15%

OptaPlanner versus traditional algorithm with domain knowledge

Average Min/Max # datasets Biggest dataset

-9%

-18%

5

2750 deliveries

55 vehicles

5 mins Late Acceptance Nearby vs First Fit Decreasing

Don't believe us? Run our open benchmarks yourself: http://www.optaplanner.org/code/benchmarks.html
Vehicle Routing Problem
Real-time planning

Warm starts to solve in milliseconds
Real-time planning

When the problem changes in real-time, the plan is adjusted in real-time.

Nightly planning  Customer visit added  Vehicles depart from depot  Customer visit added  Yellow vehicle visits customer  Customer visit added
Vehicle Routing Problem
Q & A

OptaPlanner  www.optaplanner.org
            (https://www.optaplanner.org)

Feedback    @GeoffreyDeSmet
            (https://twitter.com/GeoffreyDeSmet)