Red Hat Decision Manager 7.5

Getting started with Red Hat Business Optimizer

Last Updated: 2021-07-30

Red Hat Decision Manager 7.5 Getting started with Red Hat Business Optimizer

Red Hat Customer Content Services
brms-docs@redhat.com

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This documents describes how to start using Red Hat Business Optimizer.

Table of Contents

PREFACE 4

CHAPTER 1. INTRODUCTION TO RED HAT BUSINESS OPTIMIZER 5
1.1. PLANNING PROBLEMS 5
1.2. NP-COMPLETENESS IN PLANNING PROBLEMS 6
1.3. SOLUTIONS TO PLANNING PROBLEMS 6
1.4. CONSTRAINTS ON PLANNING PROBLEMS 7

CHAPTER 2. GETTING STARTED WITH SOLVERS IN BUSINESS CENTRAL: AN EMPLOYEE ROSTERING EXAMPLE 8
2.1. DEPLOYING THE EMPLOYEE ROSTERING SAMPLE PROJECT IN BUSINESS CENTRAL 8
2.2. RE-CREATING THE EMPLOYEE ROSTERING SAMPLE PROJECT 8
2.2.1. Setting up the employee rostering project 9
2.2.2. Problem facts and planning entities 9
2.2.3. Creating the data model for the employee rostering project 10
2.2.3.1. Creating the employee roster planning entity 11
2.2.3.2. Creating the employee roster planning solution 12
2.2.4. Employee rostering constraints 13
2.2.4.1. DRL (Drools Rule Language) rules 14
2.2.4.2. Defining constraints for employee rostering using the DRL designer 15
2.2.5. Creating rules for employee rostering using guided rules 16
2.2.5.1. Guided rules 16
2.2.5.2. Creating a guided rule to balance employee shift numbers 16
2.2.5.3. Creating a guided rule for no more than one shift per day 17
2.2.5.4. Creating a guided rule to match skills to shift requirements 18
2.2.5.5. Creating a guided rule to manage day off requests 20
2.2.6. Creating a solver configuration for employee rostering 21
2.2.7. Configuring Solver termination for the employee rostering project 21
2.3. ACCESSING THE SOLVER USING THE REST API 22
2.3.1. Registering the Solver using the REST API 22
2.3.2. Calling the Solver using the REST API 23

CHAPTER 3. GETTING STARTED WITH JAVA SOLVERS: A CLOUD BALANCING EXAMPLE 27
3.1. DOMAIN MODEL DESIGN 29
3.1.1. Designing a domain model 30
3.1.2. The Computer Class 31
3.1.3. The Process Class 31
3.1.4. The CloudBalance Class 32
3.2. RUNNING THE CLOUD BALANCING HELLO WORLD 33
3.3. SOLVER CONFIGURATION 35
3.4. SCORE CONFIGURATION 36
3.4.1. Configuring score calculation using Java 37
3.4.2. Configuring score calculation using Drools 38
3.5. FURTHER DEVELOPMENT OF THE SOLVER 40

CHAPTER 4. EXAMPLES PROVIDED WITH RED HAT BUSINESS OPTIMIZER 41
4.1. DOWNLOADING AND RUNNING THE EXAMPLES 41
4.1.1. Downloading Red Hat Business Optimizer examples 41
4.1.2. Running Business Optimizer examples 41
4.1.3. Running the Red Hat Business Optimizer examples in an IDE (IntelliJ, Eclipse, or Netbeans) 42
4.1.4. Running the web examples 43
4.2. TABLE OF BUSINESS OPTIMIZER EXAMPLES 44

4.3. N QUEENS 50
4.3.1. Domain model for N queens 52
4.4. CLOUD BALANCING 53
4.5. TRAVELING SALESMAN (TSP - TRAVELING SALESMAN PROBLEM) 54
4.6. DINNER PARTY 54
4.7. TENNIS CLUB SCHEDULING 55
4.8. MEETING SCHEDULING 56
4.9. COURSE TIMETABLING (ITC 2007 TRACK 3 - CURRICULUM COURSE SCHEDULING) 57
4.10. MACHINE REASSIGNMENT (GOOGLE ROADEF 2012) 59
4.11. VEHICLE ROUTING 62
4.11.1. Domain model for Vehicle routing 67
4.12. PROJECT JOB SCHEDULING 73
4.13. TASK ASSIGNING 75
4.14. EXAM TIMETABLING (ITC 2007 TRACK 1 - EXAMINATION) 77
4.14.1. Domain model for Exam timetabling 79
4.15. NURSE ROSTERING (INRC 2010) 80
4.16. TRAVELING TOURNAMENT PROBLEM (TTP) 85
4.17. CHEAP TIME SCHEDULING 87
4.18. INVESTMENT ASSET CLASS ALLOCATION (PORTFOLIO OPTIMIZATION) 90
4.19. CONFERENCE SCHEDULING 90
4.20. ROCK TOUR 93
4.21. FLIGHT CREW SCHEDULING 94

APPENDIX A. VERSIONING INFORMATION **95**

PREFACE

As a business rules developer, you can use the Red Hat Business Optimizer to find the optimal solution to planning problems based on a set of limited resources and under specific constraints.

Use this document to start developing solvers with Red Hat Business Optimizer.

CHAPTER 1. INTRODUCTION TO RED HAT BUSINESS OPTIMIZER

Red Hat Business Optimizer is a lightweight, embeddable planning engine that optimizes planning problems. It helps normal Java programmers solve planning problems efficiently, and it combines optimization heuristics and metaheuristics with very efficient score calculations.

For example, Red Hat Business Optimizer helps solve various use cases:

- *Employee/Patient Rosters*: It helps create timetables for nurses and keeps track of patient bed management.
- *Educational Timetables*: It helps schedule lessons, courses, exams, and conference presentations.
- *Shop Schedules*: It tracks car assembly lines, machine queue planning, and workforce task planning.
- *Cutting Stock*: It minimizes waste by reducing the consumption of resources such as paper and steel.

Every organization faces planning problems; that is, they provide products and services with a limited set of constrained resources (employees, assets, time, and money).

Red Hat Business Optimizer is open source software under the Apache Software License 2.0. It is 100% pure Java and runs on most Java virtual machines.

1.1. PLANNING PROBLEMS

A *planning problem* has an optimal goal, based on limited resources and under specific constraints. Optimal goals can be any number of things, such as:

- Maximized profits - the optimal goal results in the highest possible profit.
- Minimized ecological footprint - the optimal goal has the least amount of environmental impact.
- Maximized satisfaction for employees or customers - the optimal goal prioritizes the needs of employees or customers.

The ability to achieve these goals relies on the number of resources available. For example, the following resources might be limited:

- The number of people
- Amount of time
- Budget
- Physical assets, for example, machinery, vehicles, computers, buildings, and so on

You must also take into account the specific constraints related to these resources, such as the number of hours a person works, their ability to use certain machines, or compatibility between pieces of equipment.

Red Hat Business Optimizer helps Java programmers solve constraint satisfaction problems efficiently. It combines optimization heuristics and metaheuristics with efficient score calculation.

1.2. NP-COMPLETENESS IN PLANNING PROBLEMS

The provided use cases are *probably* NP-complete or NP-hard, which means the following statements apply:

- It is easy to verify a given solution to a problem in reasonable time.
- There is no simple way to find the optimal solution of a problem in reasonable time.

The implication is that solving your problem is probably harder than you anticipated, because the two common techniques will not suffice:

- A brute force algorithm (even a more advanced variant) will take too long.
- A quick algorithm, for example in the bin packing problem, *putting in the largest items first* will return a solution that is far from optimal.

By using advanced optimization algorithms, Business Optimizer finds a good solution in reasonable time for such planning problems.

1.3. SOLUTIONS TO PLANNING PROBLEMS

A planning problem has a number of solutions.

There are several categories of solutions:

Possible solution

A possible solution is any solution, whether or not it breaks any number of constraints. Planning problems often have an incredibly large number of possible solutions. Many of those solutions are worthless.

Feasible solution

A feasible solution is a solution that does not break any (negative) hard constraints. The number of feasible solutions tends to be relative to the number of possible solutions. Sometimes there are no feasible solutions. Every feasible solution is a possible solution.

Optimal solution

An optimal solution is a solution with the highest score. Planning problems usually have a few optimal solutions. They always have at least one optimal solution, even in the case that there are no feasible solutions and the optimal solution is not feasible.

Best solution found

The best solution found is the solution with the highest score found by an implementation in a given amount of time. The best solution found is likely to be feasible and, given enough time, it's an optimal solution.

Counterintuitively, the number of possible solutions is huge (if calculated correctly), even with a small data set.

In the examples provided in the **planner-engine** distribution folder, most instances have a vast number of possible solutions. As there is no guaranteed way to find the optimal solution, any implementation is forced to evaluate at least a subset of all those possible solutions.

Business Optimizer supports several optimization algorithms to efficiently wade through that incredibly large number of possible solutions.

Depending on the use case, some optimization algorithms perform better than others, but it is impossible to tell in advance. Using Business Optimizer, you can switch the optimization algorithm by changing the solver configuration in a few lines of XML or code.

1.4. CONSTRAINTS ON PLANNING PROBLEMS

Usually, a planning problem has at least two levels of constraints:

- A *(negative) hard constraint* must not be broken.
  For example, one teacher can not teach two different lessons at the same time.
- A *(negative) soft constraint* should not be broken if it can be avoided.
  For example, Teacher A does not like to teach on Friday afternoons.

Some problems also have positive constraints:

- A *positive soft constraint (or reward)* should be fulfilled if possible.
  For example, Teacher B likes to teach on Monday mornings.

Some basic problems only have hard constraints. Some problems have three or more levels of constraints, for example, hard, medium, and soft constraints.

These constraints define the *score calculation* (otherwise known as the *fitness function*) of a planning problem. Each solution of a planning problem can be graded with a score. With Business Optimizer, score constraints are written in an object oriented language, such as Java code or Drools rules.

This type of code is flexible and scalable.

CHAPTER 2. GETTING STARTED WITH SOLVERS IN BUSINESS CENTRAL: AN EMPLOYEE ROSTERING EXAMPLE

You can build and deploy the **employee-rostering** sample project in Business Central. The project demonstrates how to create each of the Business Central assets required to solve the shift rostering planning problem and use Red Hat Business Optimizer to find the best possible solution.

You can deploy the preconfigured **employee-rostering** project in Business Central. Alternatively, you can create the project yourself using Business Central.

NOTE

The **employee-rostering** sample project in Business Central does not include a data set. You must supply a data set in XML format using a REST API call.

2.1. DEPLOYING THE EMPLOYEE ROSTERING SAMPLE PROJECT IN BUSINESS CENTRAL

Business Central includes a number of sample projects that you can use to get familiar with the product and its features. The employee rostering sample project is designed and created to demonstrate the shift rostering use case for Red Hat Business Optimizer. Use the following procedure to deploy and run the employee rostering sample in Business Central.

Prerequisites

- Red Hat Decision Manager has been downloaded and installed. For installation options, see *Planning a Red Hat Decision Manager installation* .
- You have started Red Hat Decision Manager, as described in the installation documentation, and you are logged in to Business Central as a user with **admin** permissions.

Procedure

1. In Business Central, click **Menu** → **Design** → **Projects**.
2. In the preconfigured **MySpace** space, click **Try Samples**.
3. Select **employee-rostering** from the list of sample projects and click **Ok** in the upper-right corner to import the project.
4. After the asset list has complied, click **Build & Deploy** to deploy the employee rostering example.

The rest of this document explains each of the project assets and their configuration.

2.2. RE-CREATING THE EMPLOYEE ROSTERING SAMPLE PROJECT

The employee rostering sample project is a preconfigured project available in Business Central. You can learn about how to deploy this project in Section 2.1, "Deploying the employee rostering sample project in Business Central".

You can create the employee rostering example "from scratch". You can use the workflow in this example to create a similar project of your own in Business Central.

2.2.1. Setting up the employee rostering project

To start developing a solver in Business Central, you must set up the project.

Prerequisites

- Red Hat Decision Manager has been downloaded and installed.
- You have deployed Business Central and logged in with a user that has the **admin** role.

Procedure

1. Create a new project in Business Central by clicking **Menu → Design → Projects → Add Project**.
2. In the **Add Project** window, fill out the following fields:
   - Name: **employee-rostering**
   - **Description** (optional): Employee rostering problem optimization using Business Optimizer. Assigns employees to shifts based on their skill.

   Optionally, click **Configure Advanced Options** to populate the **Group ID**, **Artifact ID**, and **Version** information.

   - Group ID: **employeerostering**
   - Artifact ID: **employeerostering**
   - Version: **1.0.0-SNAPSHOT**
3. Click **Add** to add the project to the Business Central project repository.

2.2.2. Problem facts and planning entities

Each of the domain classes in the employee rostering planning problem can be categorized as one of the following:

- A unrelated class: not used by any of the score constraints. From a planning standpoint, this data is obsolete.
- A *problem fact* class: used by the score constraints, but does not change during planning (as long as the problem stays the same), for example, **Shift** and **Employee**. All the properties of a problem fact class are problem properties.
- A *planning entity* class: used by the score constraints and changes during planning, for example, **ShiftAssignment**. The properties that change during planning are *planning variables*. The other properties are problem properties.
  Ask yourself the following questions:
- *What class changes during planning?*
- *Which class has variables that I want the* ***Solver*** *to change?*
  That class is a planning entity.

A planning entity class needs to be annotated with the **@PlanningEntity** annotation, or defined in Business Central using the Red Hat Business Optimizer dock in the domain designer.

Each planning entity class has one or more *planning variables*, and should also have one or more defining properties.

Most use cases have only one planning entity class, and only one planning variable per planning entity class.

2.2.3. Creating the data model for the employee rostering project

Use this section to create the data objects required to run the employee rostering sample project in Business Central.

Prerequisites

- You have completed the project setup described in Section 2.2.1, "Setting up the employee rostering project".

Procedure

1. With your new project, either click **Data Object** in the project perspective, or click **Add Asset → Data Object** to create a new data object.
2. Name the first data object **Timeslot**, and select **employeerostering.employeerostering** as the **Package**.
   Click **Ok**.
3. In the **Data Objects** perspective, click **+add field** to add fields to the **Timeslot** data object.
4. In the **id** field, type **endTime**.
5. Click the drop-down menu next to **Type** and select **LocalDateTime**.
6. Click **Create and continue** to add another field.
7. Add another field with the **id startTime** and **Type LocalDateTime**.
8. Click **Create**.
9. Click **Save** in the upper-right corner to save the **Timeslot** data object.
10. Click the **x** in the upper-right corner to close the **Data Objects** perspective and return to the **Assets** menu.
11. Using the previous steps, create the following data objects and their attributes:

    Table 2.1. Skill

id	Type
name	**String**

    Table 2.2. Employee

id	Type
name	**String**
skills	**employeerostering.employeerostering.Skill[List]**

Table 2.3. Shift

id	Type
requiredSkill	**employeerostering.employeerostering.Skill**
timeslot	**employeerostering.employeerostering.Timeslot**

Table 2.4. DayOffRequest

id	Type
date	**LocalDate**
employee	**employeerostering.employeerostering.Employee**

Table 2.5. ShiftAssignment

id	Type
employee	**employeerostering.employeerostering.Employee**
shift	**employeerostering.employeerostering.Shift**

For more examples of creating data objects, see *Getting started with decision services* .

2.2.3.1. Creating the employee roster planning entity

In order to solve the employee rostering planning problem, you must create a planning entity and a solver. The planning entity is defined in the domain designer using the attributes available in the Red Hat Business Optimizer dock.

Use the following procedure to define the **ShiftAssignment** data object as the planning entity for the employee rostering example.

Prerequisites

- You have created the relevant data objects and planning entity required to run the employee rostering example by completing the procedures in Section 2.2.3, "Creating the data model for the employee rostering project".

Procedure

1. From the project **Assets** menu, open the **ShiftAssignment** data object.
2. In the **Data Objects** perspective, open the Red Hat Business Optimizer dock by clicking the on the right.
3. Select **Planning Entity**.
4. Select **employee** from the list of fields under the **ShiftAssignment** data object.
5. In the Red Hat Business Optimizer dock, select **Planning Variable**.
   In the **Value Range Id** input field, type **employeeRange**. This adds the **@ValueRangeProvider** annotation to the planning entity, which you can view by clicking the **Source** tab in the designer.

   The value range of a planning variable is defined with the **@ValueRangeProvider** annotation. A **@ValueRangeProvider** annotation always has a property **id**, which is referenced by the **@PlanningVariable** property **valueRangeProviderRefs**.
6. Close the dock and click **Save** to save the data object.

2.2.3.2. Creating the employee roster planning solution

The employee roster problem relies on a defined planning solution. The planning solution is defined in the domain designer using the attributes available in the Red Hat Business Optimizer dock.

Prerequisites

- You have created the relevant data objects and planning entity required to run the employee rostering example by completing the procedures in Section 2.2.3, "Creating the data model for the employee rostering project" and Section 2.2.3.1, "Creating the employee roster planning entity".

Procedure

1. Create a new data object with the identifier **EmployeeRoster**.
2. Create the following fields:

   Table 2.6. EmployeeRoster

id	Type
dayOffRequestList	**employeerostering.employeerostering.DayOffRequest[List]**
shiftAssignmentList	**employeerostering.employeerostering.ShiftAssignment[List]**

id	Type
shiftList	**employeerostering.employeerostering.Shift[List]**
skillList	**employeerostering.employeerostering.Skill[List]**
timeslotList	**employeerostering.employeerostering.Timeslot[List]**

3. In the **Data Objects** perspective, open the Red Hat Business Optimizer dock by clicking the

   on the right.

4. Select **Planing Solution**.

5. Leave the default **Hard soft score** as the **Solution Score Type**. This automatically generates a **score** field in the **EmployeeRoster** data object with the solution score as the type.

6. Add a new field with the following attributes:

id	Type
employeeList	**employeerostering.employeerostering.Employee[List]**

7. With the **employeeList** field selected, open the Red Hat Business Optimizer dock and select the **Planning Value Range Provider** box.
   In the **id** field, type **employeeRange**. Close the dock.

8. Click **Save** in the upper-right corner to save the asset.

2.2.4. Employee rostering constraints

Employee rostering is a planning problem. All planning problems include constraints that must be satisfied in order to find an optimal solution.

The employee rostering sample project in Business Central includes the following hard and soft constraints:

Hard constraint

- Employees are only assigned one shift per day.
- All shifts that require a particular employee skill are assigned an employee with that particular skill.

Soft constraints

- All employees are assigned a shift.

- If an employee requests a day off, their shift can be reassigned to another employee.

Hard and soft constraints can be defined in Business Central using either the free-form DRL designer, or using guided rules.

2.2.4.1. DRL (Drools Rule Language) rules

DRL (Drools Rule Language) rules are business rules that you define directly in **.drl** text files. These DRL files are the source in which all other rule assets in Business Central are ultimately rendered. You can create and manage DRL files within the Business Central interface, or create them externally as part of a Maven or Java project using Red Hat CodeReady Studio or another integrated development environment (IDE). A DRL file can contain one or more rules that define at a minimum the rule conditions (**when**) and actions (**then**). The DRL designer in Business Central provides syntax highlighting for Java, DRL, and XML.

DRL files consist of the following components:

Components in a DRL file

```
package

import

function  // Optional

query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
end

rule "rule2 name"

...
```

The following example DRL rule determines the age limit in a loan application decision service:

Example rule for loan application age limit

```
rule "Underage"
  salience 15
  agenda-group "applicationGroup"
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
```

```
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

A DRL file can contain single or multiple rules, queries, and functions, and can define resource declarations such as imports, globals, and attributes that are assigned and used by your rules and queries. The DRL package must be listed at the top of a DRL file and the rules are typically listed last. All other DRL components can follow any order.

Each rule must have a unique name within the rule package. If you use the same rule name more than once in any DRL file in the package, the rules fail to compile. Always enclose rule names with double quotation marks (**rule "rule name"**) to prevent possible compilation errors, especially if you use spaces in rule names.

All data objects related to a DRL rule must be in the same project package as the DRL file in Business Central. Assets in the same package are imported by default. Existing assets in other packages can be imported with the DRL rule.

2.2.4.2. Defining constraints for employee rostering using the DRL designer

You can create constraint definitions for the employee rostering example using the free-form DRL designer in Business Central.

Use this procedure to create a *hard constraint* where no employee can be assigned a shift that begins less than 10 hours after their previous shift ended.

Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. Click **Add Asset** → **DRL file**.
3. In the **DRL file** name field, type **ComplexScoreRules**.
4. Select the **employeerostering.employeerostering** package.
5. Click **+Ok** to create the DRL file.
6. In the **Model** tab of the DRL designer, define the **Employee10HourShiftSpace** rule as a DRL file:

```
package employeerostering.employeerostering;

rule "Employee10HourShiftSpace"
    when
        $shiftAssignment : ShiftAssignment( $employee : employee != null, $shiftEndDateTime :
shift.timeslot.endTime)
        ShiftAssignment( this != $shiftAssignment, $employee == employee, $shiftEndDateTime
<= shift.timeslot.endTime,
                $shiftEndDateTime.until(shift.timeslot.startTime,
java.time.temporal.ChronoUnit.HOURS) <10)
    then
        scoreHolder.addHardConstraintMatch(kcontext, -1);
end
```

7. Click **Save** to save the DRL file.

For more information about creating DRL files, see *Designing a decision service using DRL rules* .

2.2.5. Creating rules for employee rostering using guided rules

You can create rules that define hard and soft constraints for employee rostering using the guided rules designer in Business Central.

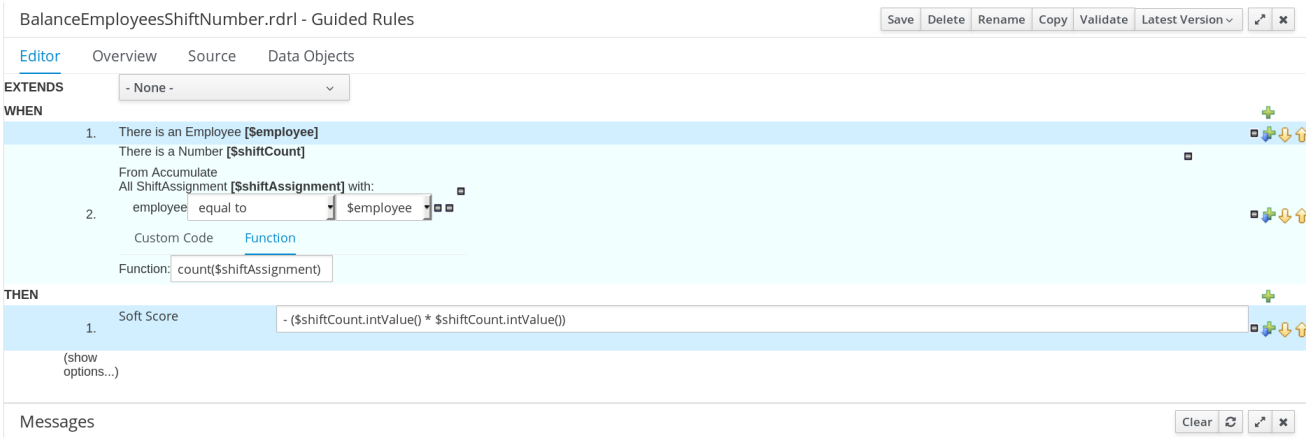
2.2.5.1. Guided rules

Guided rules are business rules that you create in a UI-based guided rules designer in Business Central that leads you through the rule-creation process. The guided rules designer provides fields and options for acceptable input based on the data objects for the rule being defined. The guided rules that you define are compiled into Drools Rule Language (DRL) rules as with all other rule assets.

All data objects related to a guided rule must be in the same project package as the guided rule. Assets in the same package are imported by default. After you create the necessary data objects and the guided rule, you can use the **Data Objects** tab of the guided rules designer to verify that all required data objects are listed or to import other existing data objects by adding a **New item**.

2.2.5.2. Creating a guided rule to balance employee shift numbers

The **BalanceEmployeesShiftNumber** guided rule creates a soft constraint that ensures shifts are assigned to employees in a way that is balanced as evenly as possible. It does this by creating a score penalty that increases when shift distribution is less even. The score formula, implemented by the rule, incentivizes the Solver to distribute shifts in a more balanced way.



Procedure

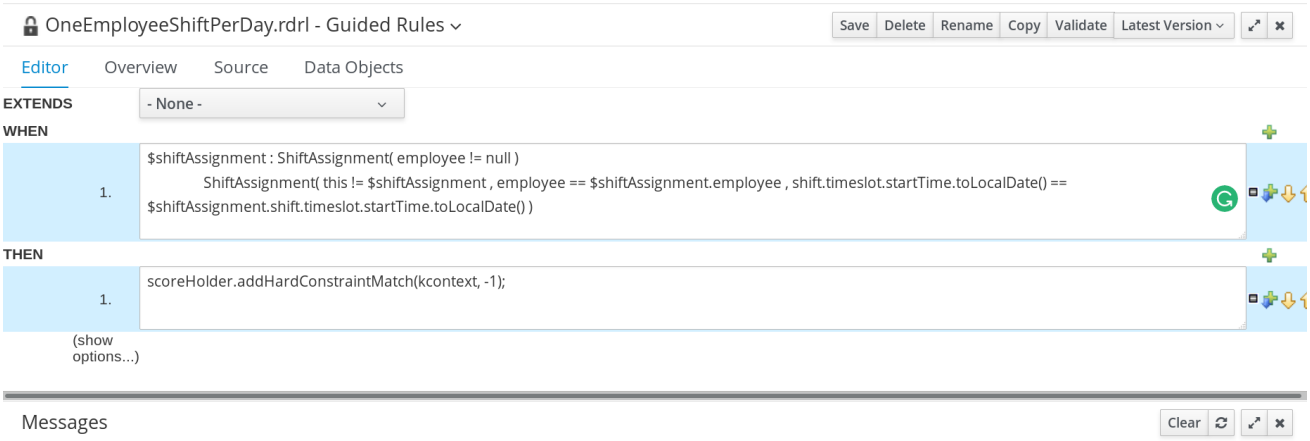
1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. Click **Add Asset** → **Guided Rule**.
3. Enter **BalanceEmployeesShiftNumber** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.
4. Click **Ok** to create the rule asset.
5. Add a **WHEN** condition by clicking the ✚ in the **WHEN** field.
6. Select **Employee** in the **Add a condition to the rule**window. Click **+Ok**.
7. Click the **Employee** condition to modify the constraints and add the variable name **$employee**.

8. Add the **WHEN** condition **From Accumulate**.
    a. Above the **From Accumulate** condition, click **click to add pattern** and select **Number** as the fact type from the drop-down list.
    b. Add the variable name **$shiftCount** to the **Number** condition.
    c. Below the **From Accumulate** condition, click **click to add pattern** and select the **ShiftAssignment** fact type from the drop-down list.
    d. Add the variable name **$shiftAssignment** to the **ShiftAssignment** fact type.
    e. Click the **ShiftAssignment** condition again and from the **Add a restriction on a field** drop-down list, select **employee**.
    f. Select **equal to** from the drop-down list next to the **employee** constraint.
    g. Click the icon next to the drop-down button to add a variable, and click **Bound variable** in the **Field value** window.
    h. Select **$employee** from the drop-down list.
    i. In the **Function** box type **count($shiftAssignment)**.
9. Add the **THEN** condition by clicking the in the **THEN** field.
10. Select **Modify Soft Score** in the **Add a new action** window. Click **+Ok**.
    a. Type the following expression into the box: **-($shiftCount.intValue()*$shiftCount.intValue())**
11. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.
12. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

2.2.5.3. Creating a guided rule for no more than one shift per day

The **OneEmployeeShiftPerDay** guided rule creates a hard constraint that employees are not assigned more than one shift per day. In the employee rostering example, this constraint is created using the guided rule designer.



Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. Click **Add Asset** → **Guided Rule**.
3. Enter **OneEmployeeShiftPerDay** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.
4. Click **Ok** to create the rule asset.
5. Add a **WHEN** condition by clicking the ✚ in the **WHEN** field.
6. Select **Free form DRL** from the **Add a condition to the rule** window.
7. In the free form DRL box, type the following condition:

```
$shiftAssignment : ShiftAssignment( employee != null )
  ShiftAssignment( this != $shiftAssignment , employee == $shiftAssignment.employee ,
shift.timeslot.startTime.toLocalDate() ==
$shiftAssignment.shift.timeslot.startTime.toLocalDate() )
```

   This condition states that a shift cannot be assigned to an employee that already has another shift assignment on the same day.

8. Add the **THEN** condition by clicking the ✚ in the **THEN** field.
9. Select **Add free form DRL** from the **Add a new action** window.
10. In the free form DRL box, type the following condition:

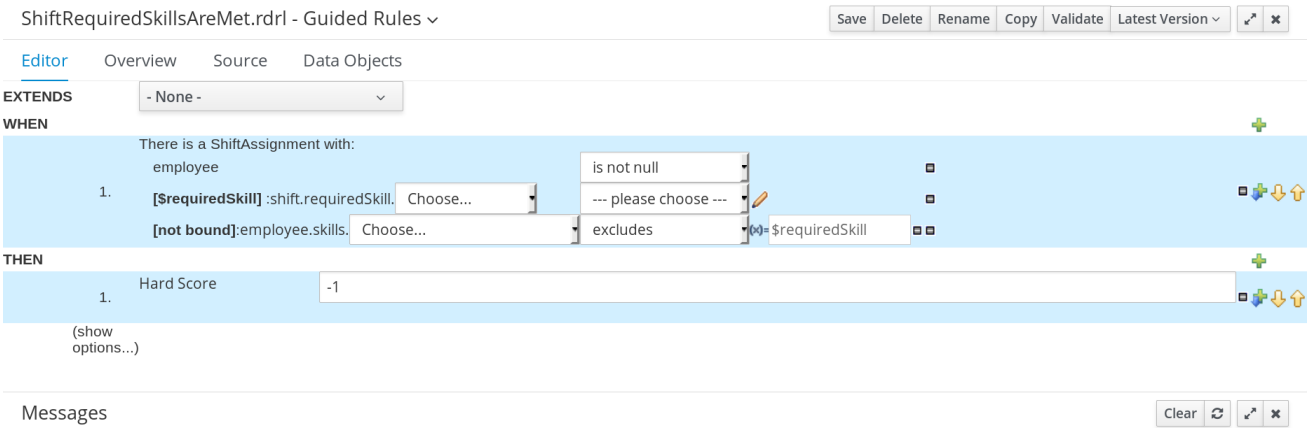
```
scoreHolder.addHardConstraintMatch(kcontext, -1);
```

11. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.
12. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

2.2.5.4. Creating a guided rule to match skills to shift requirements

The **ShiftReqiredSkillsAreMet** guided rule creates a hard constraint that ensures all shifts are assigned an employee with the correct set of skills. In the employee rostering example, this constraint is created using the guided rule designer.



Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. Click **Add Asset** → **Guided Rule**.
3. Enter **ShiftReqiredSkillsAreMet** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.
4. Click **Ok** to create the rule asset.
5. Add a **WHEN** condition by clicking the ✚ in the **WHEN** field.
6. Select **ShiftAssignment** in the **Add a condition to the rule**window. Click **+Ok**.
7. Click the **ShiftAssignment** condition, and select **employee** from the **Add a restriction on a field** drop-down list.
8. In the designer, click the drop-down list next to **employee** and select **is not null**.
9. Click the **ShiftAssignment** condition, and click **Expression editor**.
   a. In the designer, click **[not bound]** to open the **Expression editor**, and bind the expression to the variable **$requiredSkill**. Click **Set**.
   b. In the designer, next to **$requiredSkill**, select **shift** from the first drop-down list, then **requiredSkill** from the next drop-down list.
10. Click the **ShiftAssignment** condition, and click **Expression editor**.
    a. In the designer, next to **[not bound]**, select **employee** from the first drop-down list, then **skills** from the next drop-down list.
    b. Leave the next drop-down list as **Choose**.
    c. In the next drop-down box, change **please choose** to **excludes**.
    d. Click the icon next to **excludes**, and in the **Field value** window, click the **New formula** button.

e. Type **$requiredSkill** into the formula box.

11. Add the **THEN** condition by clicking the ✚ in the **THEN** field.

12. Select **Modify Hard Score** in the **Add a new action** window. Click **+Ok**.

13. Type **-1** into the score actions box.

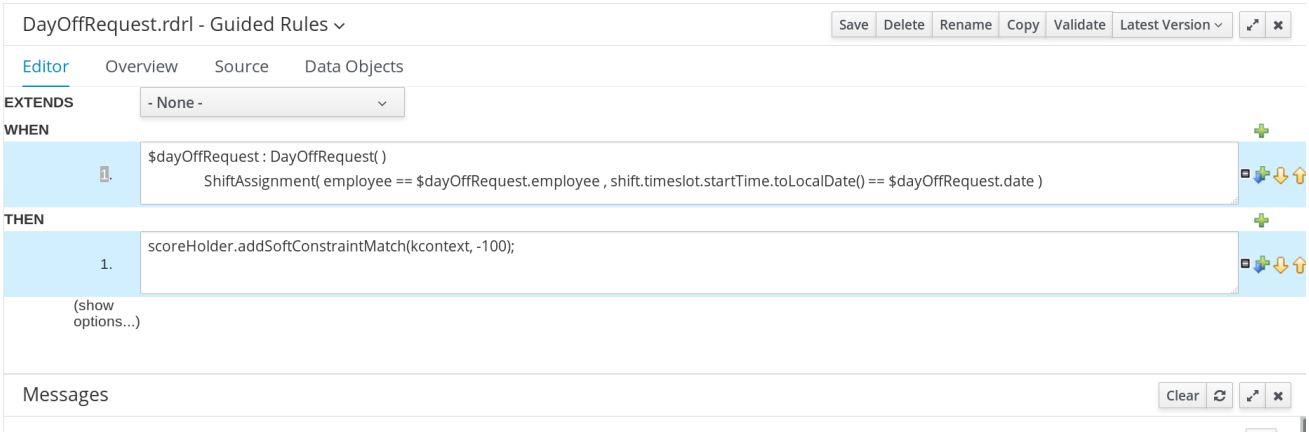
14. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.

15. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

2.2.5.5. Creating a guided rule to manage day off requests

The **DayOffRequest** guided rule creates a soft constraint. This constraint allows a shift to be reassigned to another employee in the event the employee who was originally assigned the shift is no longer able to work that day. In the employee rostering example, this constraint is created using the guided rule designer.



Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.

2. Click **Add Asset** → **Guided Rule**.

3. Enter **DayOffRequest** as the **Guided Rule** name and select the **employeerostering.employeerostering** Package.

4. Click **Ok** to create the rule asset.

5. Add a **WHEN** condition by clicking the ✚ in the **WHEN** field.

6. Select **Free form DRL** from the **Add a condition to the rule** window.

7. In the free form DRL box, type the following condition:

```
$dayOffRequest : DayOffRequest( )
  ShiftAssignment( employee == $dayOffRequest.employee ,
shift.timeslot.startTime.toLocalDate() == $dayOffRequest.date )
```

This condition states if a shift is assigned to an employee who has made a day off request, the employee can be unassigned the shift on that day.

8. Add the **THEN** condition by clicking the in the **THEN** field.
9. Select **Add free form DRL** from the **Add a new action** window.
10. In the free form DRL box, type the following condition:

```
scoreHolder.addSoftConstraintMatch(kcontext, -100);
```

11. Click **Validate** in the upper-right corner to check all rule conditions are valid. If the rule validation fails, address any problems described in the error message, review all components in the rule, and try again to validate the rule until the rule passes.
12. Click **Save** to save the rule.

For more information about creating guided rules, see *Designing a decision service using guided rules* .

2.2.6. Creating a solver configuration for employee rostering

You can create and edit Solver configurations in Business Central. The Solver configuration designer creates a solver configuration that can be run after the project is deployed.

Prerequisites

- Red Hat Decision Manager has been downloaded and installed.
- You have created and configured all of the relevant assets for the employee rostering example.

Procedure

1. In Business Central, click **Menu → Projects**, and click your project to open it.
2. In the **Assets** perspective, click **Add Asset → Solver configuration**
3. In the **Create new Solver configuration** window, type the name **EmployeeRosteringSolverConfig** for your Solver and click **Ok**.
   This opens the **Solver configuration** designer.
4. In the **Score Director Factory** configuration section, define a KIE base that contains scoring rule definitions. The employee rostering sample project uses **defaultKieBase**.
   a. Select one of the KIE sessions defined within the KIE base. The employee rostering sample project uses **defaultKieSession**.
5. Click **Validate** in the upper-right corner to check the **Score Director Factory** configuration is correct. If validation fails, address any problems described in the error message, and try again to validate until the configuration passes.
6. Click **Save** to save the Solver configuration.

2.2.7. Configuring Solver termination for the employee rostering project

You can configure the Solver to terminate after a specified amount of time. By default, the planning engine is given an unlimited time period to solve a problem instance.

The employee rostering sample project is set up to run for 30 seconds.

Prerequisites

- You have created all relevant assets for the employee rostering project and created the **EmployeeRosteringSolverConfig** solver configuration in Business Central as described in Section 2.2.6, "Creating a solver configuration for employee rostering".

Procedure

1. Open the **EmployeeRosteringSolverConfig** from the **Assets** perspective. This will open the **Solver configuration** designer.
2. In the **Termination** section, click **Add** to create new termination element within the selected logical group.
3. Select the **Time spent** termination type from the drop-down list. This is added as an input field in the termination configuration.
4. Use the arrows next to the time elements to adjust the amount of time spent to 30 seconds.
5. Click **Validate** in the upper-right corner to check the **Score Director Factory** configuration is correct. If validation fails, address any problems described in the error message, and try again to validate until the configuration passes.
6. Click **Save** to save the Solver configuration.

2.3. ACCESSING THE SOLVER USING THE REST API

After deploying or re-creating the sample solver, you can access it using the REST API.

You must register a solver instance using the REST API. Then you can supply data sets and retrieve optimized solutions.

Prerequisites

- The employee rostering project is set up and deployed according to the previous sections in this document. You can either deploy the sample project, as described in Section 2.1, "Deploying the employee rostering sample project in Business Central", or re-create the project, as described in Section 2.2, "Re-creating the employee rostering sample project".

2.3.1. Registering the Solver using the REST API

You must register the solver instance using the REST API before you can use the solver.

Each solver instance is capable of optimizing one planning problem at a time.

Procedure

1. Create a HTTP request using the following header:

```
authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml
```

2. Register the Solver using the following request:

   PUT

   http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver

   Request body

```
<solver-instance>
  <solver-config-
file>employeerostering/employeerostering/EmployeeRosteringSolverConfig.solver.xml</s
olver-config-file>
</solver-instance>
```

2.3.2. Calling the Solver using the REST API

After registering the solver instance, you can use the REST API to submit a data set to the solver and to retrieve an optimized solution.

Procedure

1. Create a HTTP request using the following header:

```
authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml
```

2. Submit a request to the Solver with a data set, as in the following example:

   POST

   http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/state/solving

   Request body

```
<employeerostering.employeerostering.EmployeeRoster>
  <employeeList>
    <employeerostering.employeerostering.Employee>
      <name>John</name>
      <skills>
        <employeerostering.employeerostering.Skill>
          <name>reading</name>
        </employeerostering.employeerostering.Skill>
      </skills>
    </employeerostering.employeerostering.Employee>
    <employeerostering.employeerostering.Employee>
      <name>Mary</name>
      <skills>
```

```
      <employeerostering.employeerostering.Skill>
        <name>writing</name>
      </employeerostering.employeerostering.Skill>
    </skills>
  </employeerostering.employeerostering.Employee>
  <employeerostering.employeerostering.Employee>
    <name>Petr</name>
    <skills>
      <employeerostering.employeerostering.Skill>
        <name>speaking</name>
      </employeerostering.employeerostering.Skill>
    </skills>
  </employeerostering.employeerostering.Employee>
</employeeList>
<shiftList>
  <employeerostering.employeerostering.Shift>
    <timeslot>
      <startTime>2017-01-01T00:00:00</startTime>
      <endTime>2017-01-01T01:00:00</endTime>
    </timeslot>
    <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee/skills/emplo
yeerostering.employeerostering.Skill"/>
  </employeerostering.employeerostering.Shift>
  <employeerostering.employeerostering.Shift>
    <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
    <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emp
loyeerostering.employeerostering.Skill"/>
  </employeerostering.employeerostering.Shift>
  <employeerostering.employeerostering.Shift>
    <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
    <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emp
loyeerostering.employeerostering.Skill"/>
  </employeerostering.employeerostering.Shift>
</shiftList>
<skillList>
  <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee/skills/employe
erostering.employeerostering.Skill"/>
  <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emplo
yeerostering.employeerostering.Skill"/>
  <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emplo
yeerostering.employeerostering.Skill"/>
</skillList>
<timeslotList>
  <employeerostering.employeerostering.Timeslot
reference="../../shiftList/employeerostering.employeerostering.Shift/timeslot"/>
</timeslotList>
<dayOffRequestList/>
<shiftAssignmentList>
  <employeerostering.employeerostering.ShiftAssignment>
    <shift reference="../../../shiftList/employeerostering.employeerostering.Shift"/>
```

```
    </employeerostering.employeerostering.ShiftAssignment>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift[3]"/>
    </employeerostering.employeerostering.ShiftAssignment>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift[2]"/>
    </employeerostering.employeerostering.ShiftAssignment>
  </shiftAssignmentList>
</employeerostering.employeerostering.EmployeeRoster>
```

3. Request the best solution to the planning problem:

GET

http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/bestsolution

Example response

```
<solver-instance>
  <container-id>employee-rostering</container-id>
  <solver-id>solver1</solver-id>
  <solver-config-
file>employeerostering/employeerostering/EmployeeRosteringSolverConfig.solver.xml</s
olver-config-file>
  <status>NOT_SOLVING</status>
  <score
scoreClass="org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore">0hard/0soft<
/score>
  <best-solution class="employeerostering.employeerostering.EmployeeRoster">
    <employeeList>
      <employeerostering.employeerostering.Employee>
        <name>John</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>reading</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
      <employeerostering.employeerostering.Employee>
        <name>Mary</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>writing</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
      <employeerostering.employeerostering.Employee>
        <name>Petr</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>speaking</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
```

```
    </employeeList>
    <shiftList>
      <employeerostering.employeerostering.Shift>
        <timeslot>
          <startTime>2017-01-01T00:00:00</startTime>
          <endTime>2017-01-01T01:00:00</endTime>
        </timeslot>
        <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee/skills/emplo
yeerostering.employeerostering.Skill"/>
      </employeerostering.employeerostering.Shift>
      <employeerostering.employeerostering.Shift>
        <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
        <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emp
loyeerostering.employeerostering.Skill"/>
      </employeerostering.employeerostering.Shift>
      <employeerostering.employeerostering.Shift>
        <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
        <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emp
loyeerostering.employeerostering.Skill"/>
      </employeerostering.employeerostering.Shift>
    </shiftList>
    <skillList>
      <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee/skills/employe
erostering.employeerostering.Skill"/>
      <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emplo
yeerostering.employeerostering.Skill"/>
      <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emplo
yeerostering.employeerostering.Skill"/>
    </skillList>
    <timeslotList>
      <employeerostering.employeerostering.Timeslot
reference="../../shiftList/employeerostering.employeerostering.Shift/timeslot"/>
    </timeslotList>
    <dayOffRequestList/>
    <shiftAssignmentList/>
    <score>0hard/0soft</score>
  </best-solution>
</solver-instance>
```

CHAPTER 3. GETTING STARTED WITH JAVA SOLVERS: A CLOUD BALANCING EXAMPLE

An example demonstrates development of a basic Red Hat Business Optimizer solver using Java code.

Suppose your company owns a number of cloud computers and needs to run a number of processes on those computers. You must assign each process to a computer.

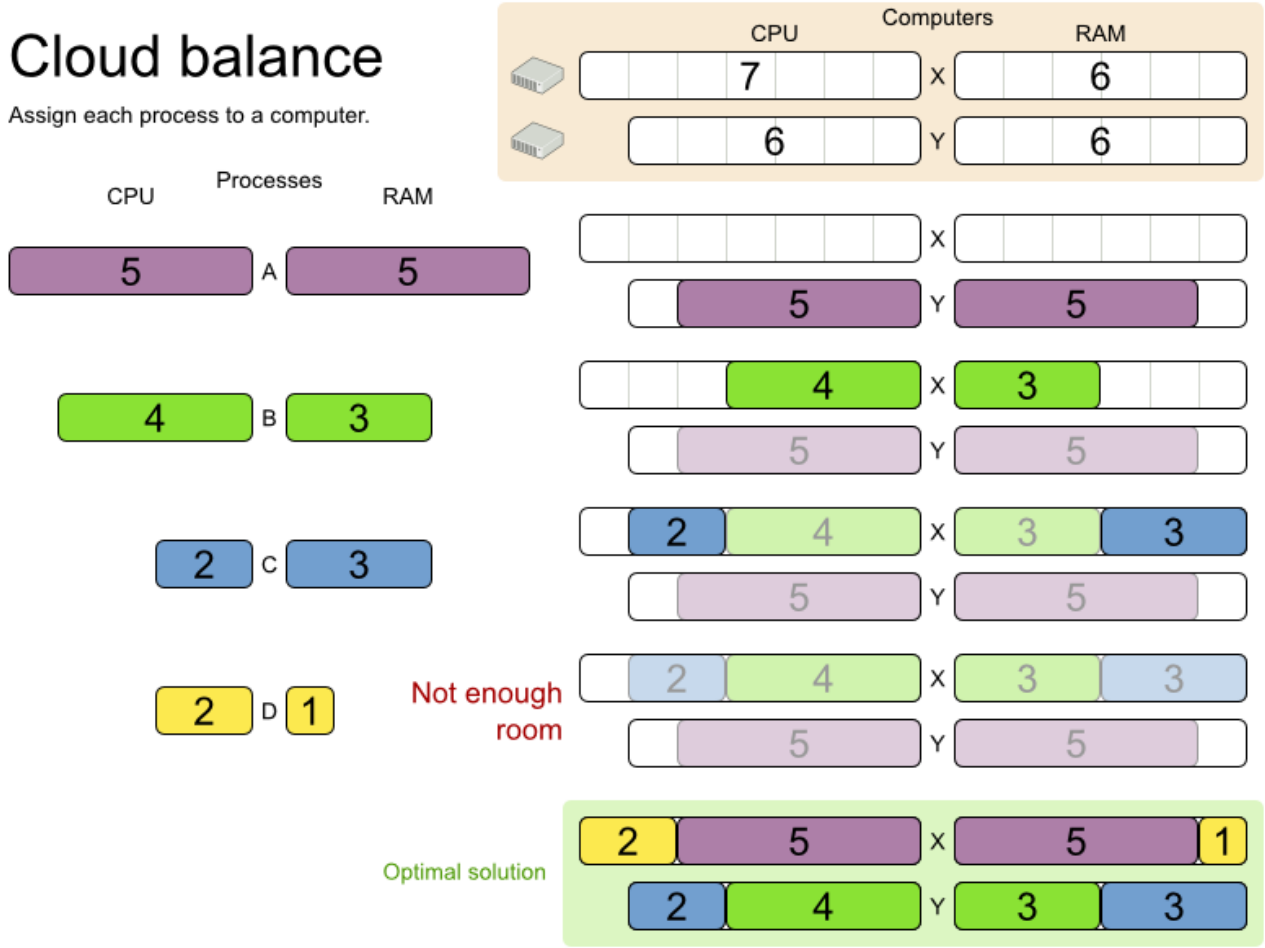
The following hard constraints must be fulfilled:

- Every computer must be able to handle the minimum hardware requirements of the sum of its processes:
  - **CPU capacity**: The CPU power of a computer must be at least the sum of the CPU power required by the processes assigned to that computer.
  - **Memory capacity**: The RAM memory of a computer must be at least the sum of the RAM memory required by the processes assigned to that computer.
  - **Network capacity**: The network bandwidth of a computer must be at least the sum of the network bandwidth required by the processes assigned to that computer.

The following soft constraints should be optimized:

- Each computer that has one or more processes assigned incurs a maintenance cost (which is fixed per computer).
  - **Cost**: Minimize the total maintenance cost.

This problem is a form of *bin packing*. In the following simplified example, we assign four processes to two computers with two constraints (CPU and RAM) with a simple algorithm:



The simple algorithm used here is the *First Fit Decreasing* algorithm, which assigns the bigger processes first and assigns the smaller processes to the remaining space. As you can see, it is not optimal, as it does not leave enough room to assign the yellow process **D**.

Business Optimizer finds a more optimal solution by using additional, smarter algorithms. It also scales: both in data (more processes, more computers) and constraints (more hardware requirements, other constraints).

The following summary applies to this example, as well as to an advanced implementation with more constraints that is described in Section 4.10, “Machine reassignment (Google ROADEF 2012)” :

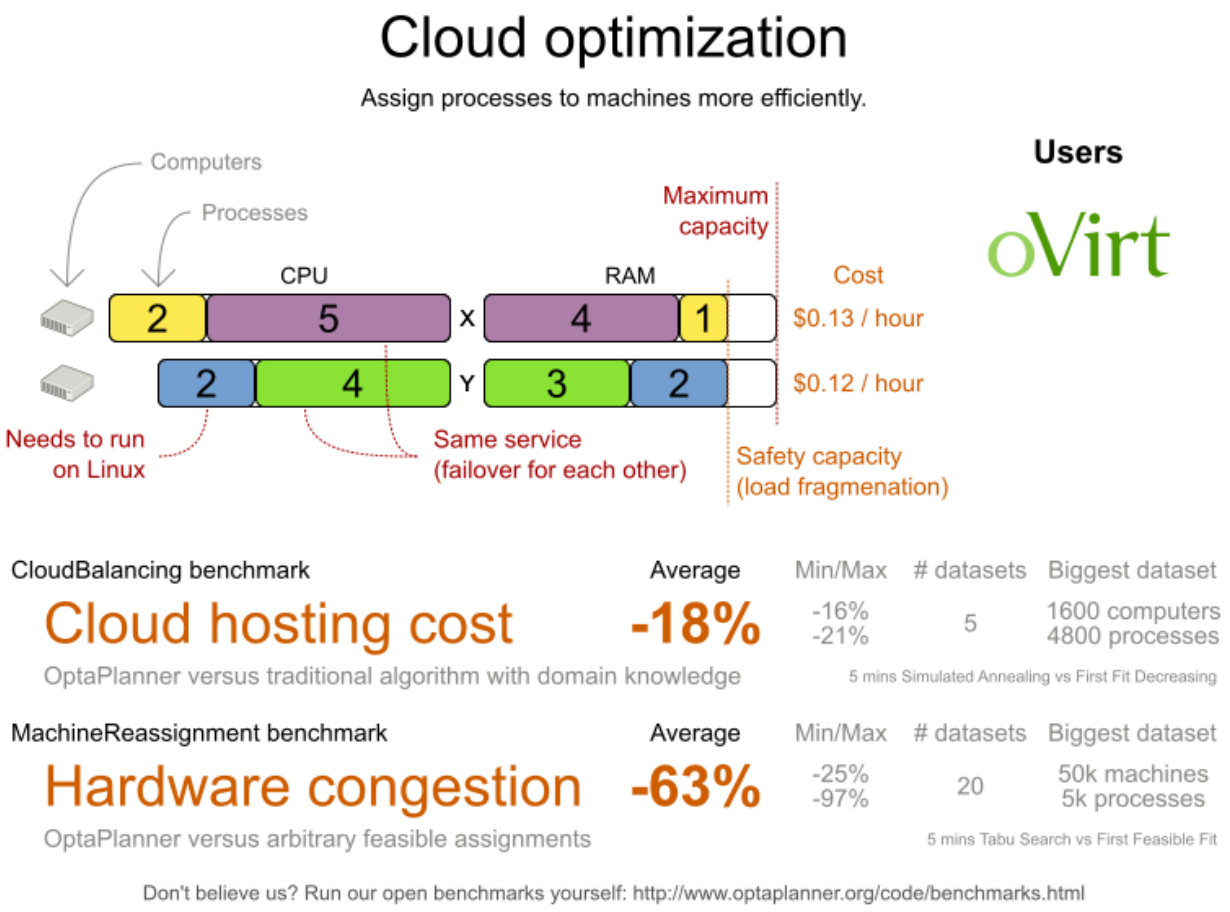


Table 3.1. Cloud balancing problem size

Problem size	Computers	Processes	Search space
2computers-6processes	2	6	64
3computers-9processes	3	9	10^4
4computers-012processes	4	12	10^7
100computers-300processes	100	300	10^600
200computers-600processes	200	600	10^1380
400computers-1200processes	400	1200	10^3122
800computers-2400processes	800	2400	10^6967

3.1. DOMAIN MODEL DESIGN

Using a *domain model* helps determine which classes are planning entities and which of their properties are planning variables. It also helps to simplify contraints, improve performance, and increase flexibility for future needs.

3.1.1. Designing a domain model

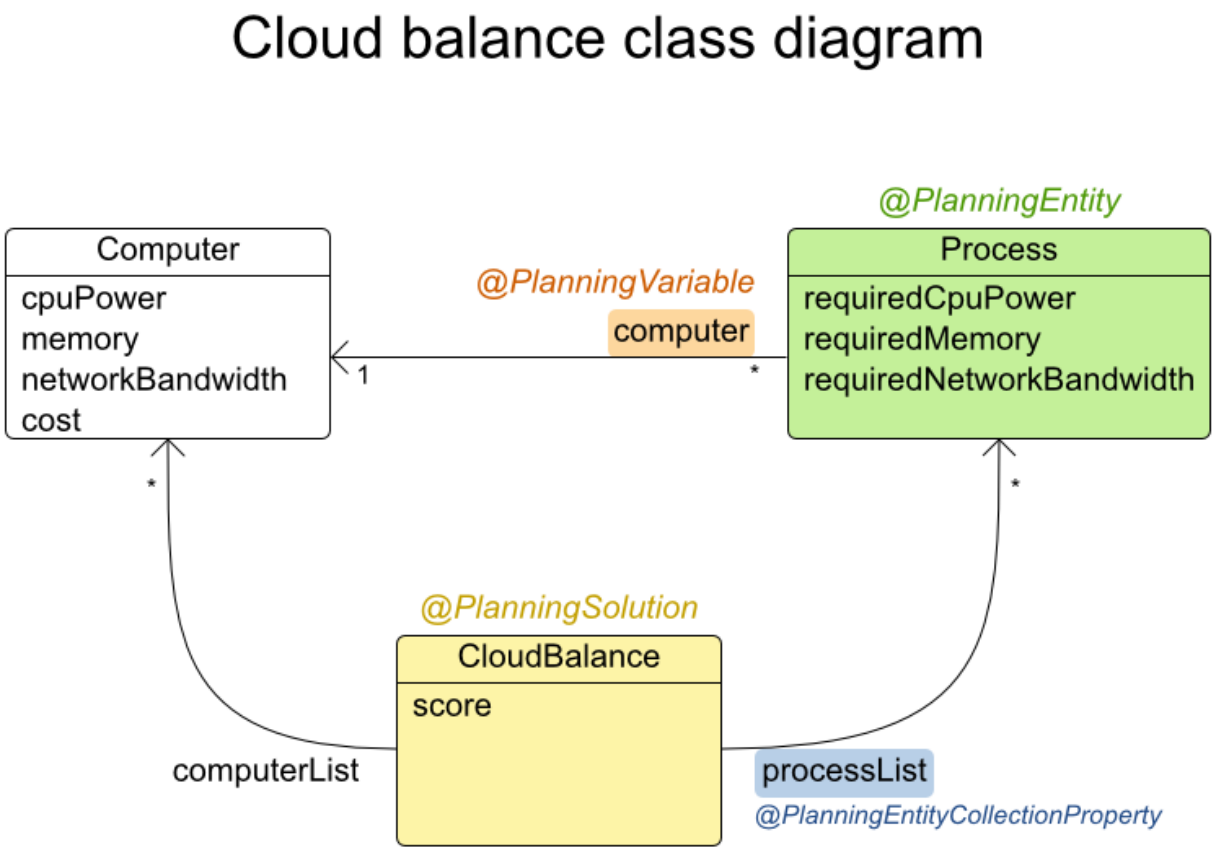
To create a domain model, define all the objects that represent the input data for the problem. In this example, the objects are processes and computers.

A separate object in the domain model must represent a full data set of the problem, which contains the input data as well as a solution. In this example, this object holds a list of computers and a list of processes. Each process is assigned to a computer; the distribution of processes between computers is the solution.

Procedure

1. Draw a class diagram of your domain model.
2. Normalize it to remove duplicate data.
3. Write down some *sample instances* for each class. Sample instances are entity properties that are relevant for planning purposes.
   - **Computer**: Represents a computer with certain hardware and maintenance costs.
     In this example, the sample instances for the **Computer** class are **cpuPower**, **memory**, **networkBandwidth**, **cost**.
   - **Process**: Represents a process with a demand. Needs to be assigned to a **Computer** by Planner.
     Sample instances for **Process** are **requiredCpuPower**, **requiredMemory**, and **requiredNetworkBandwidth**.
   - **CloudBalance**: Represents the distribution of processes between computers. Contains every **Computer** and **Process** for a certain data set.
     For an object representing the full data set and solution, a sample instance holding the *score* must be present. Business Optimizer can calculate and compare the scores for different solutions; the solution with the highest score is the optimal solution. Therefore, the sample instance for **CloudBalance** is **score**.
4. Determine which relationships (or fields) change during planning:
   - *Planning entity*: The class (or classes) that Business Optimizer can change during solving. In this example, it is the class **Process**, because we can move processes to different computers.
     - A class representing input data that Business Optimizer can not change is known as a *problem fact*.
   - *Planning variable*: The property (or properties) of a planning entity class that changes during solving. In this example, it is the property **computer** on the class **Process**.
   - *Planning solution*: The class that represents a solution to the problem. This class must represent the full data set and contain all planning entities. In this example that is the class **CloudBalance**.

In the UML class diagram below, the Business Optimizer concepts are already annotated:



You can find the class definitions for this example in the **examples/sources/src/main/java/org/optaplanner/examples/cloudbalancing/domain** directory.

3.1.2. The Computer Class

The **Computer** class is a Java object that stores data, sometimes known as a POJO (Plain Old Java Object). Usually, you will have more of this kind of classes with input data.

Example 3.1. CloudComputer.java

```
public class CloudComputer ... {

    private int cpuPower;
    private int memory;
    private int networkBandwidth;
    private int cost;

    ... // getters
}
```

3.1.3. The Process Class

The **Process** class is the class that is modified during solving.

We need to tell Business Optimizer that it can change the property **computer**. To do this, annotate the class with **@PlanningEntity** and annotate the **getComputer()** getter with **@PlanningVariable**.

Of course, the property **computer** needs a setter too, so Business Optimizer can change it during solving.

Example 3.2. CloudProcess.java

```
@PlanningEntity(...)
public class CloudProcess ... {

    private int requiredCpuPower;
    private int requiredMemory;
    private int requiredNetworkBandwidth;

    private CloudComputer computer;

    ... // getters

    @PlanningVariable(valueRangeProviderRefs = {"computerRange"})
    public CloudComputer getComputer() {
        return computer;
    }

    public void setComputer(CloudComputer computer) {
        computer = computer;
    }

    // ************************************************************************
    // Complex methods
    // ************************************************************************

    ...

}
```

Business Optimizer needs to know which values it can choose from to assign to the property **computer**. Those values are retrieved from the method **CloudBalance.getComputerList()** on the planning solution, which returns a list of all computers in the current data set.

The **@PlanningVariable**'s **valueRangeProviderRefs** parameter on **CloudProcess.getComputer()** needs to match with the **@ValueRangeProvider**'s **id** on **CloudBalance.getComputerList()**.

NOTE

You can also use annotations on fields instead of getters.

3.1.4. The CloudBalance Class

The **CloudBalance** class has a *@PlanningSolution* annotation.

This class holds a list of all computers and processes. It represents both the planning problem and (if it is initialized) the planning solution.

The **CloudBalance** class has the following key attributes:

- It holds a collection of processes that Business Optimizer can change. We annotate the getter **getProcessList()** with **@PlanningEntityCollectionProperty**, so that Business Optimizer can retrieve the processes that it can change. To save a solution, Business Optimizer initializes a new instance of the class with the list of changed processes.

  1. It also has a **@PlanningScore** annotated property **score**, which is the **Score** of that solution in its current state. Business Optimizer automatically updates it when it calculates a **Score** for a solution instance; therefore, this property needs a setter.

  2. Especially for score calculation with Drools, the property **computerList** needs to be annotated with a **@ProblemFactCollectionProperty** so that Business Optimizer can retrieve a list of computers (problem facts) and make it available to the decision engine.

Example 3.3. CloudBalance.java

```
@PlanningSolution
public class CloudBalance ... {

    private List<CloudComputer> computerList;

    private List<CloudProcess> processList;

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    public List<CloudComputer> getComputerList() {
        return computerList;
    }

    @PlanningEntityCollectionProperty
    public List<CloudProcess> getProcessList() {
        return processList;
    }

    @PlanningScore
    public HardSoftScore getScore() {
        return score;
    }

    public void setScore(HardSoftScore score) {
        this.score = score;
    }

    ...
}
```

3.2. RUNNING THE CLOUD BALANCING HELLO WORLD

You can run a sample "hello world" application to demonstrate the solver.

Procedure

1. Download and configure the examples in your preferred IDE. For instructions on downloading and configiring examples in an IDE, see Section 4.1.3, "Running the Red Hat Business Optimizer examples in an IDE (IntelliJ, Eclipse, or Netbeans)".

2. Create a run configuration with the following main class:
   org.optaplanner.examples.cloudbalancing.app.CloudBalancingHelloWorld
   By default, the Cloud Balancing Hello World is configured to run for 120 seconds.

Result

The application executes the following code:

Example 3.4. CloudBalancingHelloWorld.java

```
public class CloudBalancingHelloWorld {

  public static void main(String[] args) {
    // Build the Solver
    SolverFactory<CloudBalance> solverFactory =
SolverFactory.createFromXmlResource("org/optaplanner/examples/cloudbalancing/solver/cloudBal
ancingSolverConfig.xml");
    Solver<CloudBalance> solver = solverFactory.buildSolver();

    // Load a problem with 400 computers and 1200 processes
    CloudBalance unsolvedCloudBalance = new
CloudBalancingGenerator().createCloudBalance(400, 1200);

    // Solve the problem
    CloudBalance solvedCloudBalance = solver.solve(unsolvedCloudBalance);

    // Display the result
    System.out.println("\nSolved cloudBalance with 400 computers and 1200 processes:\n" +
toDisplayString(solvedCloudBalance));
  }

  ...
}
```

The code example does the following:

1. Build the **Solver** based on a solver configuration (in this case an XML file, **cloudBalancingSolverConfig.xml**, from the classpath).
   Building the **Solver** is the most complicated part of this procedure. For more details, see Section 3.3, "Solver Configuration".

   ```
   SolverFactory<CloudBalance> solverFactory = SolverFactory.createFromXmlResource(

"org/optaplanner/examples/cloudbalancing/solver/cloudBalancingSolverConfig.xml");
   Solver solver<CloudBalance> = solverFactory.buildSolver();
   ```

2. Load the problem.

CloudBalancingGenerator generates a random problem: you will replace this with a class that loads a real problem, for example from a database.

```
        CloudBalance unsolvedCloudBalance = new
CloudBalancingGenerator().createCloudBalance(400, 1200);
```

3. Solve the problem.

```
        CloudBalance solvedCloudBalance = solver.solve(unsolvedCloudBalance);
```

4. Display the result.

```
        System.out.println("\nSolved cloudBalance with 400 computers and 1200 processes:\n"
            + toDisplayString(solvedCloudBalance));
```

3.3. SOLVER CONFIGURATION

The solver configuration file determines how the solving process works; it is considered a part of the code. The file is named **examples/sources/src/main/resources/org/optaplanner/examples/cloudbalancing/solver/cloudBalancingSolverConfig.xml**.

Example 3.5. cloudBalancingSolverConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  <!-- Domain model configuration -->
  <scanAnnotatedClasses/>

  <!-- Score configuration -->
  <scoreDirectorFactory>

<easyScoreCalculatorClass>org.optaplanner.examples.cloudbalancing.optional.score.CloudBalanci
ngEasyScoreCalculator</easyScoreCalculatorClass>
    <!--
<scoreDrl>org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl</scoreDr
>-->
  </scoreDirectorFactory>

  <!-- Optimization algorithms configuration -->
  <termination>
    <secondsSpentLimit>30</secondsSpentLimit>
  </termination>
</solver>
```

This solver configuration consists of three parts:

1. **Domain model configuration**: *What can Business Optimizer change?*
   We need to make Business Optimizer aware of our domain classes. In this configuration, it will automatically scan all classes in your classpath (for a **@PlanningEntity** or **@PlanningSolution** annotation):

```
<scanAnnotatedClasses/>
```

2. **Score configuration**: *How should Business Optimizer optimize the planning variables? What is our goal?*
   Since we have hard and soft constraints, we use a **HardSoftScore**. But we need to tell Business Optimizer how to calculate the score, depending on our business requirements. Further down, we will look into two alternatives to calculate the score: using a basic Java implementation and using Drools DRL.

```
 <scoreDirectorFactory>

<easyScoreCalculatorClass>org.optaplanner.examples.cloudbalancing.optional.score.CloudBal
ancingEasyScoreCalculator</easyScoreCalculatorClass>
   <!--
<scoreDrl>org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl</sco
eDrl>-->
 </scoreDirectorFactory>
```

3. **Optimization algorithms configuration**: *How should Business Optimizer optimize it?* In this case, we use the default optimization algorithms (because no explicit optimization algorithms are configured) for 30 seconds:

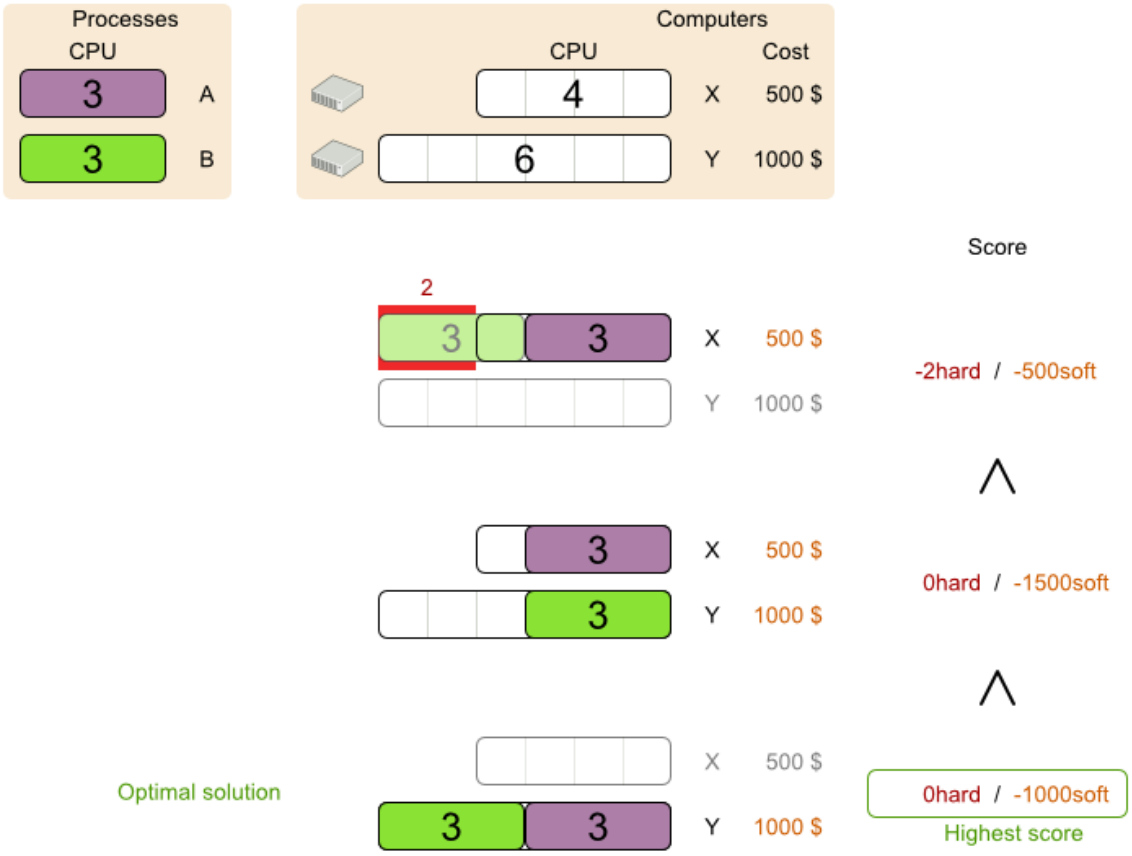
```
<termination>
  <secondsSpentLimit>30</secondsSpentLimit>
</termination>
```

   Business Optimizer should get a good result in seconds (and even in less than 15 milliseconds if the real-time planning feature is used), but the more time it has, the better the result will be. Advanced use cases might use different termination criteria than a hard time limit.

   The default algorithms will already easily surpass human planners and most in-house implementations. You can use the advanced Benchmarker feature to power tweak to get even better results.

3.4. SCORE CONFIGURATION

Business Optimizer will search for the **Solution** with the highest **Score**. This example uses a **HardSoftScore**, which means Business Optimizer will look for the solution with no hard constraints broken (fulfill hardware requirements) and as little as possible soft constraints broken (minimize maintenance cost).



Of course, Business Optimizer needs to be told about these domain-specific score constraints. You can define constraints using the Java or Drools languages.

3.4.1. Configuring score calculation using Java

One way to define a score function is to implement the interface **EasyScoreCalculator** in plain Java.

Procedure

1. In the **cloudBalancingSolverConfig.xml** file, add or uncomment the setting:

```
<scoreDirectorFactory>

<easyScoreCalculatorClass>org.optaplanner.examples.cloudbalancing.optional.score.CloudBalancingEasyScoreCalculator</easyScoreCalculatorClass>
</scoreDirectorFactory>
```

2. Implement the **calculateScore(Solution)** method to return a **HardSoftScore** instance.

Example 3.6. CloudBalancingEasyScoreCalculator.java

```
public class CloudBalancingEasyScoreCalculator implements
EasyScoreCalculator<CloudBalance> {

    /**
     * A very simple implementation. The double loop can easily be removed by using
Maps as shown in
```

```
     * {@link
CloudBalancingMapBasedEasyScoreCalculator#calculateScore(CloudBalance)}.
     */
    public HardSoftScore calculateScore(CloudBalance cloudBalance) {
        int hardScore = 0;
        int softScore = 0;
        for (CloudComputer computer : cloudBalance.getComputerList()) {
            int cpuPowerUsage = 0;
            int memoryUsage = 0;
            int networkBandwidthUsage = 0;
            boolean used = false;

            // Calculate usage
            for (CloudProcess process : cloudBalance.getProcessList()) {
                if (computer.equals(process.getComputer())) {
                    cpuPowerUsage += process.getRequiredCpuPower();
                    memoryUsage += process.getRequiredMemory();
                    networkBandwidthUsage += process.getRequiredNetworkBandwidth();
                    used = true;
                }
            }

            // Hard constraints
            int cpuPowerAvailable = computer.getCpuPower() - cpuPowerUsage;
            if (cpuPowerAvailable < 0) {
                hardScore += cpuPowerAvailable;
            }
            int memoryAvailable = computer.getMemory() - memoryUsage;
            if (memoryAvailable < 0) {
                hardScore += memoryAvailable;
            }
            int networkBandwidthAvailable = computer.getNetworkBandwidth() -
networkBandwidthUsage;
            if (networkBandwidthAvailable < 0) {
                hardScore += networkBandwidthAvailable;
            }

            // Soft constraints
            if (used) {
                softScore -= computer.getCost();
            }
        }
        return HardSoftScore.valueOf(hardScore, softScore);
    }

}
```

Even if we optimize the code above to use **Map**s to iterate through the **processList** only once, *it is still slow* because it does not do incremental score calculation.

To fix that, either use incremental Java score calculation or Drools score calculation. Incremental Java score calculation is not covered in this guide.

3.4.2. Configuring score calculation using Drools

You can use Drools rule language (DRL) to define constraints. Drools score calculation uses incremental calculation, where every score constraint is written as one or more score rules.

Using the decision engine for score calculation enables you to integrate with other Drools technologies, such as decision tables (XLS or web based), Business Central, and other supported features.

Procedure

1. Add a **scoreDrl** resource in the classpath to use the decision engine as a score function. In the **cloudBalancingSolverConfig.xml** file, add or uncomment the setting:

```
  <scoreDirectorFactory>

<scoreDrl>org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl</s
coreDrl>
  </scoreDirectorFactory>
```

2. Create the hard constraints. These constraints ensure that all computers have enough CPU, RAM and network bandwidth to support all their processes:

Example 3.7. cloudBalancingScoreRules.drl - Hard Constraints

```
...

import org.optaplanner.examples.cloudbalancing.domain.CloudBalance;
import org.optaplanner.examples.cloudbalancing.domain.CloudComputer;
import org.optaplanner.examples.cloudbalancing.domain.CloudProcess;

global HardSoftScoreHolder scoreHolder;

//
##########################################################################
####
// Hard constraints
//
##########################################################################
####

rule "requiredCpuPowerTotal"
    when
        $computer : CloudComputer($cpuPower : cpuPower)
        accumulate(
            CloudProcess(
                computer == $computer,
                $requiredCpuPower : requiredCpuPower);
            $requiredCpuPowerTotal : sum($requiredCpuPower);
            $requiredCpuPowerTotal > $cpuPower
        )
    then
        scoreHolder.addHardConstraintMatch(kcontext, $cpuPower -
$requiredCpuPowerTotal);
end

rule "requiredMemoryTotal"
    ...
end
```

```
rule "requiredNetworkBandwidthTotal"
    ...
end
```

3. Create a soft constraint. This constraint miminizes the maintenance cost. It is applied only if hard constraints are met:

Example 3.8. cloudBalancingScoreRules.drl - Soft Constraints

```
//
##########################################################################
####
// Soft constraints
//
##########################################################################
####

rule "computerCost"
    when
        $computer : CloudComputer($cost : cost)
        exists CloudProcess(computer == $computer)
    then
        scoreHolder.addSoftConstraintMatch(kcontext, - $cost);
end
```

3.5. FURTHER DEVELOPMENT OF THE SOLVER

Now that this example works, you can try developing it further. For example, you can enrich the domain model and add extra constraints such as these:

- Each **Process** belongs to a **Service**. A computer might crash, so processes running the same service should (or must) be assigned to different computers.
- Each **Computer** is located in a **Building**. A building might burn down, so processes of the same services should (or must) be assigned to computers in different buildings.

CHAPTER 4. EXAMPLES PROVIDED WITH RED HAT BUSINESS OPTIMIZER

Several Red Hat Business Optimizer examples are shipped with Red Hat Decision Manager. You can review the code for examples and modify it as necessary to suit your needs.

NOTE

Red Hat does not provide support for the example code included in the Red Hat Decision Manager distribution.

4.1. DOWNLOADING AND RUNNING THE EXAMPLES

You can download the Red Hat Business Optimizer examples from the Red Hat Software Downloads website and run them.

4.1.1. Downloading Red Hat Business Optimizer examples

You can download the examples as a part of the Red Hat Decision Manager add-ons package.

Procedure

1. Download the **rhdm-7.5.1-add-ons.zip** file from the Software Downloads page.
2. Decompress the file.
3. Decompress the **rhdm-7.5-planner-engine.zip** file from the decompressed directory.

Result

In the decompressed **rhdm-7.5-planner-engine** directory, you can find example source code under the following subdirectories: * **examples/sources/src/main/java/org/optaplanner/examples** * **examples/sources/src/main/resources/org/optaplanner/examples** * **webexamples/sources/src/main/java/org/optaplanner/examples** * **webexamples/sources/src/main/resources/org/optaplanner/examples**

The table of examples in Section 4.2, “Table of Business Optimizer examples” lists directory names that are used for individual examples.

4.1.2. Running Business Optimizer examples

Red Hat Business Optimizer includes a number of examples to demonstrate a variety of use cases.

Prerequisites

- You have downloaded and decompressed the examples. For instructions about these actions, see Section 4.1.1, “Downloading Red Hat Business Optimizer examples” .

Procedure

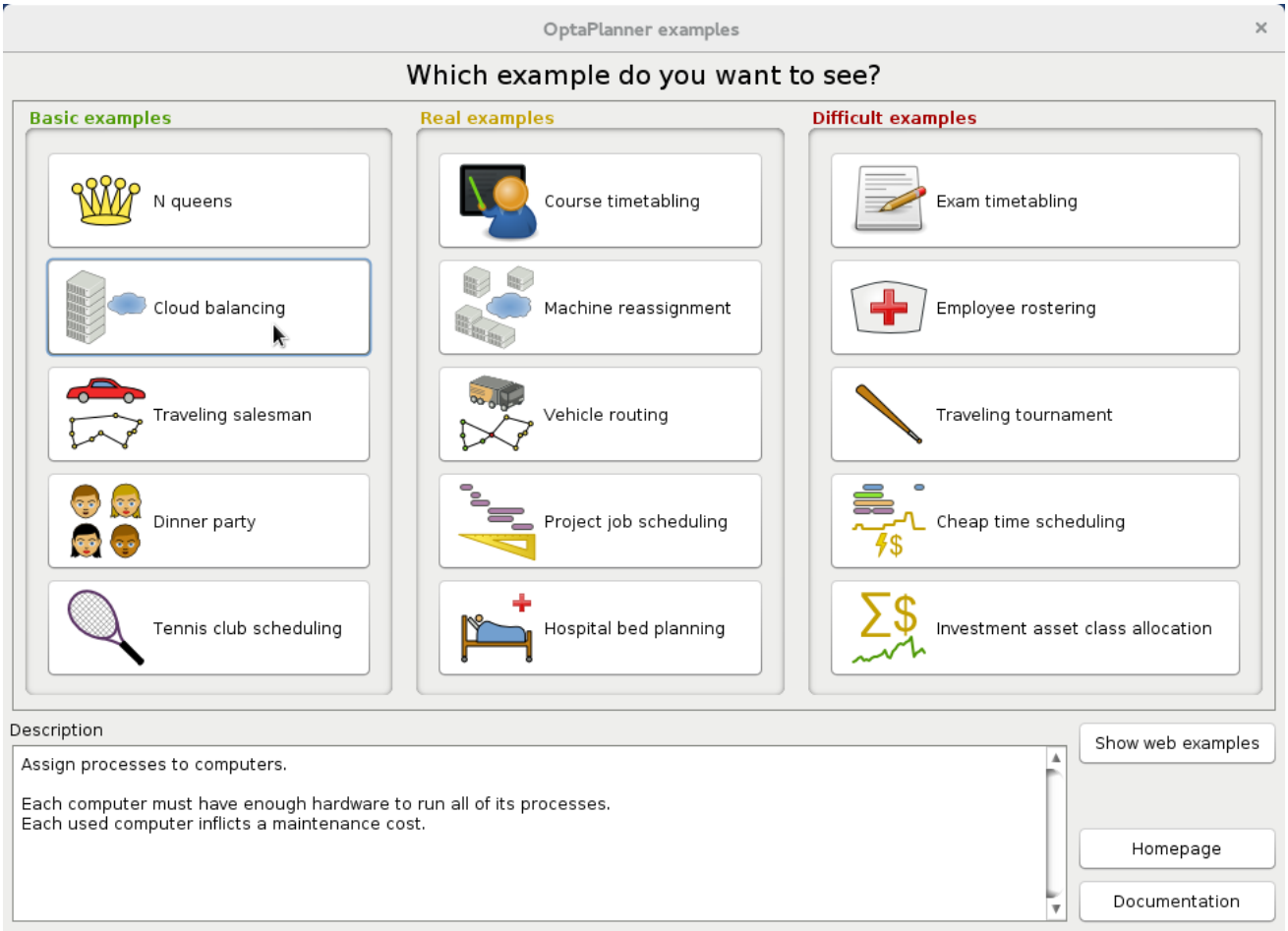
1. In the **rhdm-7.5.1-planner-engine** folder, open the **examples** directory and use the appropriate script to run the examples:
   Linux or Mac:
   -

```
$ cd examples
$ ./runExamples.sh
```

Windows:

```
$ cd examples
$ runExamples.bat
```

Select and run an example from the GUI application window:



NOTE

Red Hat Business Optimizer itself has no GUI dependencies. It runs just as well on a server or a mobile JVM as it does on the desktop.

4.1.3. Running the Red Hat Business Optimizer examples in an IDE (IntelliJ, Eclipse, or Netbeans)

If you use an integrated development environment (IDE), such as IntelliJ, Eclipse, or Netbeans, you can run your downloaded Red Hat Business Optimizer examples within your development environment.

Prerequisites

- You have downloaded and extracted the examples. For instructions about these actions, see Section 4.1.1, “Downloading Red Hat Business Optimizer examples” .

Procedure

1. Open the Red Hat Business Optimizer examples as a new project:
   a. For IntelliJ or Netbeans, open **examples/sources/pom.xml** as the new project. The Maven integration guides you through the rest of the installation; skip the rest of the steps in this procedure.
   b. For Eclipse, open a new project for the directory **examples/sources**.
2. Add all the JARs to the classpath from the directory **binaries** and the directory **examples/binaries** , except for the **examples/binaries/optaplanner-examples-*.jar** file.
3. Add the Java source directory **src/main/java** and the Java resources directory **src/main/resources**.
4. Create a run configuration:
   - Main class: **org.optaplanner.examples.app.OptaPlannerExamplesApp**
   - VM parameters (optional): **-Xmx512M -server -Dorg.optaplanner.examples.dataDir=examples/sources/data**
   - Working directory: **examples/sources**
5. Run the run configuration.

4.1.4. Running the web examples

Besides the GUI examples, Red Hat Decision Manager also includes a set of web examples for Red Hat Business Optimizer. The web examples include:

- Vehicle routing: Calculating the shortest possible route to pick up all items required for a number of different customers using either Leaflet or Google Maps visualizations.
- Cloud balancing: Assigning processes across computers with different specifications and costs.

Prerequisites

- You have downloaded and extracted the Red Hat Business Optimizer examples from the Red Hat Decision Manager add-ons package. For instructions, see Section 4.1.1, “Downloading Red Hat Business Optimizer examples”.

The web examples require several JEE APIs to run, such as the following APIs:

- Servlet
- JAX-RS
- CDI

These APIs are not required for Business Optimizer itself.

Procedure

1. Download a JEE application server, such as JBoss EAP or WildFly and unzip it.

2. In the decompressed **rhdm-7.5.1-planner-engine** directory, open the subdirectory **webexamples/binaries** and deploy the **optaplanner-webexamples-*.war** file on the JEE application server.
   If using JBoss EAP in standalone mode, this can be done by adding the **optaplanner-webexamples-*.war** file to the **JBOSS_home/standalone/deployments** folder.

3. Open the following address in a web browser: http://localhost:8080/optaplanner-webexamples/.

4.2. TABLE OF BUSINESS OPTIMIZER EXAMPLES

Some of the Business Optimizer examples solve problems that are presented in academic contests. The **Contest** column in the following table lists the contests. It also identifies an example as being either *realistic* or *unrealistic* for the purpose of a contest. A *realistic contest* is *an official, independent contest*:

A *realistic contest* is an official, independent contest that meets the following standards:

- Clearly defined real-world use cases
- Real-world constraints
- Multiple real-world datasets
- Reproducible results within a specific time limit on specific hardware
- Serious participation from the academic and/or enterprise Operations Research community.

Realistic contests provide an objective comparison of Business Optimizer with competitive software and academic research.

Table 4.1. Examples overview

Example	Domain	Size	Contest	Directory name
N queens	• 1 entity class ○ 1 variable	• Entity ⇐ **256** • Value ⇐ **256** • Search space ⇐ **10^616**	• Pointless (cheatable)	**nqueens**
Cloud balancing	• 1 entity class ○ 1 variable	• Entity ⇐ **2400** • Value ⇐ **800** • Search space ⇐ **10^6967**	• No • Defined by us	**cloudbalancing**

Example	Domain	Size	Contest	Directory name
Traveling salesman	• 1 entity class ○ 1 chained variable	• Entity ⇐ **980** • Value ⇐ **980** • Search space ⇐ **10^2504**	• Unrealistic • TSP web	**tsp**
Dinner party	• 1 entity class ○ 1 variable	• Entity ⇐ **144** • Value ⇐ **72** • Search space ⇐ **10^310**	• Unrealistic	**dinnerParty**
Tennis club scheduling	• 1 entity class ○ 1 variable	• Entity ⇐ **72** • Value ⇐ **7** • Search space ⇐ **10^60**	• No • Defined by us	**tennis**
Meeting scheduling	• 1 entity class ○ 2 variables	• Entity ⇐ **10** • Value ⇐ **320** and ⇐ **5** • Search space ⇐ **10^320**	• No • Defined by us	**meetingscheduling**

Example	Domain	Size	Contest	Directory name
Course timetabling	• 1 entity class ○ 2 variables	• Entity ⇐ **434** • Value ⇐ **25** and ⇐ **20** • Search space ⇐ **10^1171**	• Realistic • ITC 2007 track 3	**curriculumCourse**
Machine reassignment	• 1 entity class ○ 1 variable	• Entity ⇐ **50000** • Value ⇐ **5000** • Search space ⇐ **10^184948**	• Nearly realistic • ROADEF 2012	**machineReassignment**

Vehicle routing	• 1 entity class
	  ○ 1 chained variable
	• 1 shadow entity class
	  ○ 1 automatic shadow variable
		• Entity ⇐ **2740**
		• Value ⇐ **2795**
		• Search space ⇐ **10^8380**
			• Unrealistic
			• VRP web
				vehiclerouting

Example	Domain	Size	Contest	Directory name
Vehicle routing with time windows	• All of Vehicle routing • 1 shadow variable	• Entity ⇐ **2740** • Value ⇐ **2795** • Search space ⇐ **10^8380**	• Unrealistic • VRP web	**vehiclerouting**

Project job scheduling	• 1 entity class
	  ○ 2 variables
	  ○ 1 shadow variable
		• Entity ⇐ **640**
		• Value ⇐ **?** and ⇐ **?**
		• Search space ⇐ **?**
			• Nearly realistic
			• MISTA 2013
				projectjobscheduling

Task assigning	• 1 entity class
	  ○ 1 chained variable
	  ○ 1 shadow variable
	• 1 shadow entity class
	  ○ 1 automatic shadow variable
		• Entity ⇐ **500**
		• Value ⇐ **520**
		• Search space ⇐ **10^1168**
			• No
			• Defined by us
				taskassigning

Example	Domain	Size	Contest	Directory name
Exam timetabling	• 2 entity classes (same hierarchy) ○ 2 variables	• Entity ⇐ **1096** • Value ⇐ **80** and ⇐ **49** • Search space ⇐ **10^3374**	• Realistic • ITC 2007 track 1	• **examination**
Nurse rostering	• 1 entity class ○ 1 variable	• Entity ⇐ **752** • Value ⇐ **50** • Search space ⇐ **10^1277**	• Realistic • INRC 2010	**nurserostering**
Traveling tournament	• 1 entity class ○ 1 variable	• Entity ⇐ **1560** • Value ⇐ **78** • Search space ⇐ **10^2301**	• Unrealistic • TTP	**travelingtournament**
Cheap time scheduling	• 1 entity class ○ 2 variables	• Entity ⇐ **500** • Value ⇐ **100** and ⇐ **288** • Search space ⇐ **10^20078**	• Nearly realistic • ICON Energy	**cheaptimescheduling**

Example	Domain	Size	Contest	Directory name
Investment	• 1 entity class • 1 variable	• Entity ⇐ **11** • Value = **1000** • Search space ⇐ **10^4**	• No • Defined by us	**investment**
Conference scheduling	• 1 entity class ○ 2 variables	• Entity ⇐ **216** • Value ⇐ **18** and ⇐ **20** • Search space ⇐ **10^552**	• No • Defined by us	**conferencescheduling**

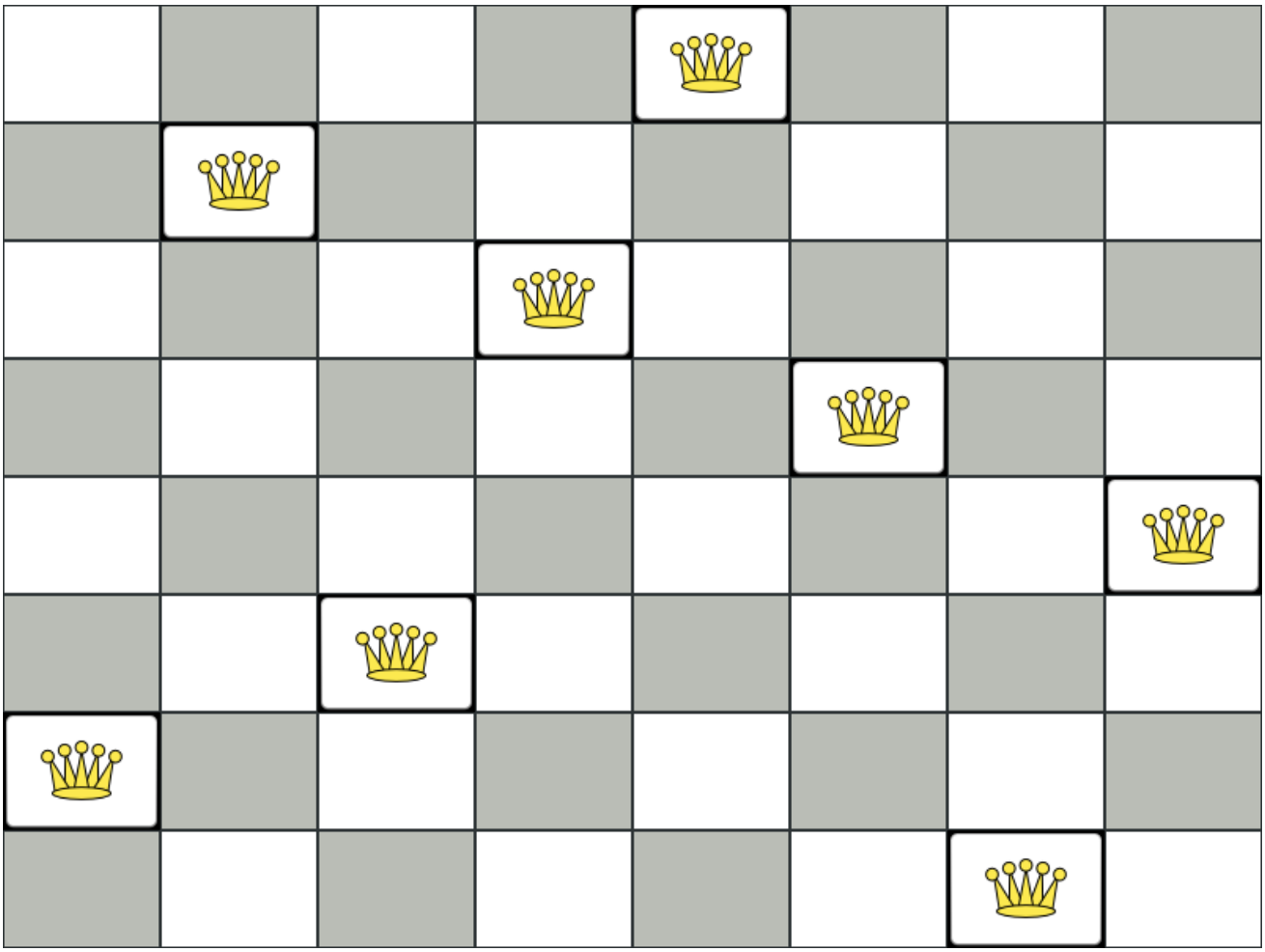
Rock tour	• 1 entity class
	  ○ 1 chained variable
	  ○ 4 shadow variables
	• 1 shadow entity class
	  ○ 1 automatic shadow variable
		• Entity ⇐ **47**
		• Value ⇐ **48**
		• Search space ⇐ **10^59**
			• No
			• Defined by us
				rocktour

Example	Domain	Size	Contest	Directory name

Flight crew scheduling	• 1 entity class
	  ○ 1 variable
	• 1 shadow entity class
	  ○ 1 automatic shadow variable
		• Entity ⇐ **4375**
		• Value ⇐ **750**
		• Search space ⇐ **10^12578**
			• No
			• Defined by us
				flightcrewscheduling

4.3. N QUEENS

Place *n* queens on a *n* sized chessboard so that no two queens can attack each other. The most common *n* queens puzzle is the eight queens puzzle, with $n = 8$:



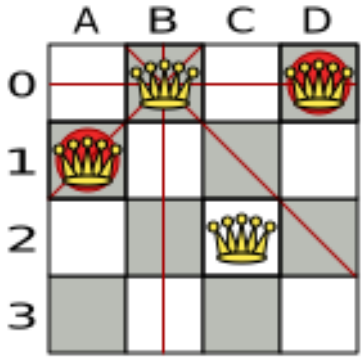
Constraints:

- Use a chessboard of *n* columns and *n* rows.
- Place *n* queens on the chessboard.
- No two queens can attack each other. A queen can attack any other queen on the same horizontal, vertical or diagonal line.

This documentation heavily uses the four queens puzzle as the primary example.

A proposed solution could be:

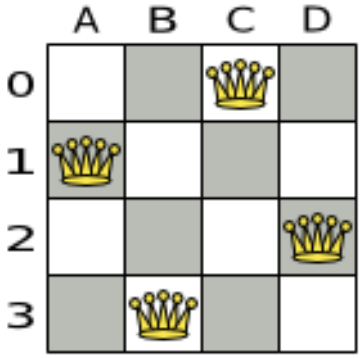
Figure 4.1. A wrong solution for the Four queens puzzle



The above solution is wrong because queens **A1** and **B0** can attack each other (so can queens **B0** and **D0**). Removing queen **B0** would respect the "no two queens can attack each other" constraint, but would break the "place *n* queens" constraint.

Below is a correct solution:

Figure 4.2. A correct solution for the Four queens puzzle



All the constraints have been met, so the solution is correct.

Note that most *n* queens puzzles have multiple correct solutions. We will focus on finding a single correct solution for a given *n*, not on finding the number of possible correct solutions for a given *n*.

Problem size

```
4queens   has   4 queens with a search space of    256.
8queens   has   8 queens with a search space of   10^7.
16queens  has  16 queens with a search space of  10^19.
```

```
32queens  has  32 queens with a search space of  10^48.
64queens  has  64 queens with a search space of 10^115.
256queens has 256 queens with a search space of 10^616.
```

The implementation of the *n* queens example has not been optimized because it functions as a beginner example. Nevertheless, it can easily handle 64 queens. With a few changes it has been shown to easily handle 5000 queens and more.

4.3.1. Domain model for N queens

This example uses the domain model to solve the four queens problem.

- **Creating a Domain Model**
  A good domain model will make it easier to understand and solve your planning problem.

  This is the domain model for the *n* queens example:

```
public class Column {

    private int index;

    // ... getters and setters
}
```

```
public class Row {

    private int index;

    // ... getters and setters
}
```

```
public class Queen {

    private Column column;
    private Row row;

    public int getAscendingDiagonalIndex() {...}
    public int getDescendingDiagonalIndex() {...}

    // ... getters and setters
}
```

- **Calculating the Search Space.**
  A **Queen** instance has a **Column** (for example: 0 is column A, 1 is column B, ...) and a **Row** (its row, for example: 0 is row 0, 1 is row 1, ...).

  The ascending diagonal line and the descending diagonal line can be calculated based on the column and the row.

  The column and row indexes start from the upper left corner of the chessboard.

```
public class NQueens {

    private int n;
```

```
    private List<Column> columnList;
    private List<Row> rowList;

    private List<Queen> queenList;

    private SimpleScore score;

    // ... getters and setters
}
```

1. **Finding the Solution**
   A single **NQueens** instance contains a list of all **Queen** instances. It is the **Solution** implementation which will be supplied to, solved by, and retrieved from the Solver.

Notice that in the four queens example, NQueens's **getN()** method will always return four.

Figure 4.3. A solution for Four Queens

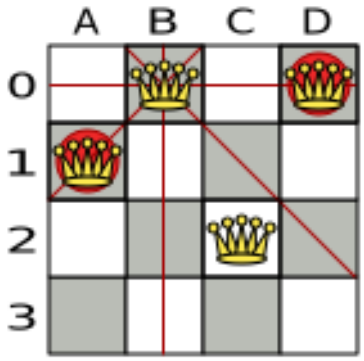


Table 4.2. Details of the solution in the domain model

	columnIndex	rowIndex	ascendingDiagonalIndex (columnIndex + rowIndex)	descendingDiagonalIndex (columnIndex - rowIndex)
A1	0	1	**1 (**)**	-1
B0	1	**0 (*)**	**1 (**)**	1
C2	2	2	4	0
D0	3	**0 (*)**	3	3

When two queens share the same column, row or diagonal line, such as (*) and (**), they can attack each other.

4.4. CLOUD BALANCING

For information about this example, see Chapter 3, *Getting started with Java solvers: A cloud balancing example*.

4.5. TRAVELING SALESMAN (TSP - TRAVELING SALESMAN PROBLEM)

Given a list of cities, find the shortest tour for a salesman that visits each city exactly once.

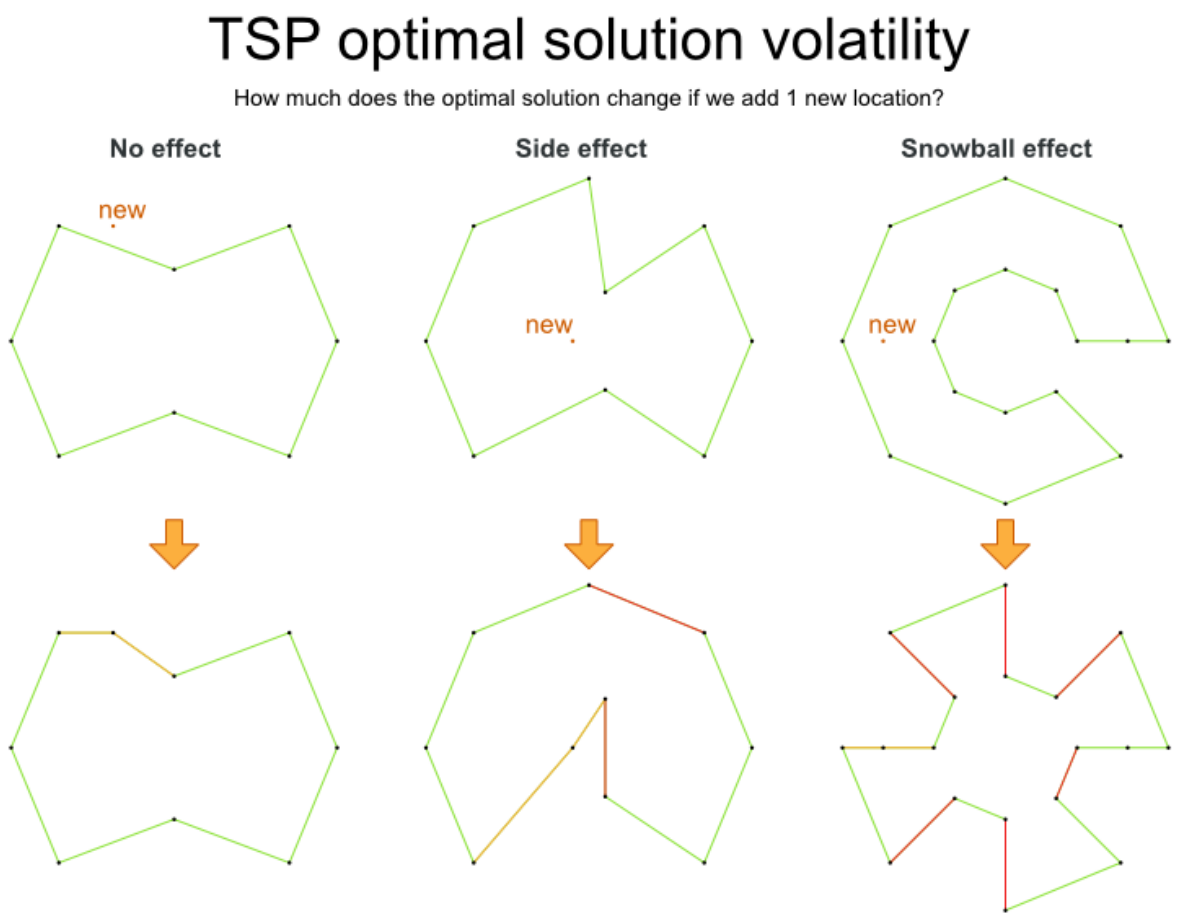
The problem is defined by Wikipedia. It is one of the most intensively studied problems in computational mathematics. Yet, in the real world, it is often only part of a planning problem, along with other constraints, such as employee shift rostering constraints.

Problem size

```
dj38     has  38 cities with a search space of   10^43.
europe40 has  40 cities with a search space of   10^46.
st70     has  70 cities with a search space of   10^98.
pcb442   has 442 cities with a search space of  10^976.
lu980    has 980 cities with a search space of 10^2504.
```

Problem difficulty

Despite TSP's simple definition, the problem is surprisingly hard to solve. Because it is an NP-hard problem (like most planning problems), the optimal solution for a specific problem dataset can change a lot when that problem dataset is slightly altered:



4.6. DINNER PARTY

Miss Manners is throwing another dinner party.

- This time she invited 144 guests and prepared 12 round tables with 12 seats each.
- Every guest should sit next to someone (left and right) of the opposite gender.
- And that neighbour should have at least one hobby in common with the guest.
- At every table, there should be two politicians, two doctors, two socialites, two coaches, two teachers and two programmers.
- And the two politicians, two doctors, two coaches and two programmers should not be the same kind at a table.

Drools Expert also has the normal Miss Manners example (which is much smaller) and employs an exhaustive heuristic to solve it. Planner's implementation is far more scalable because it uses heuristics to find the best solution and Drools Expert to calculate the score of each solution.

Problem size

> wedding01 has 18 jobs, 144 guests, 288 hobby practicians, 12 tables and 144 seats with a search space of 10^310.

4.7. TENNIS CLUB SCHEDULING

Every week the tennis club has four teams playing round robin against each other. Assign those four spots to the teams fairly.

Hard constraints:

- Conflict: A team can only play once per day.
- Unavailability: Some teams are unavailable on some dates.

Medium constraints:

- Fair assignment: All teams should play an (almost) equal number of times.

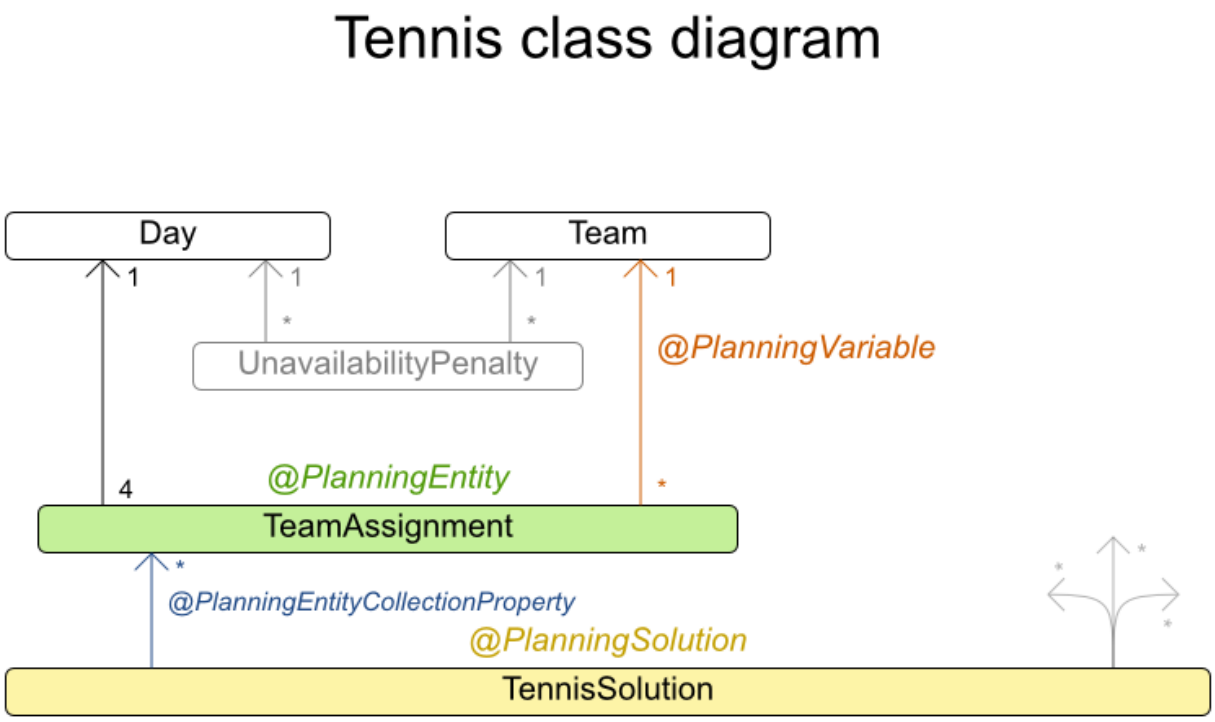
Soft constraints:

- Evenly confrontation: Each team should play against every other team an equal number of times.

Problem size

> munich-7teams has 7 teams, 18 days, 12 unavailabilityPenalties and 72 teamAssignments with a search space of 10^60.

Figure 4.4. Domain model



4.8. MEETING SCHEDULING

Assign each meeting to a starting time and a room. Meetings have different durations.

Hard constraints:

- Room conflict: two meetings must not use the same room at the same time.
- Required attendance: A person cannot have two required meetings at the same time.
- Required room capacity: A meeting must not be in a room that doesn't fit all of the meeting's attendees.
- Start and end on same day: A meeting shouldn't be scheduled over multiple days.

Medium constraints:

- Preferred attendance: A person cannot have two preferred meetings at the same time, nor a preferred and a required meeting at the same time.

Soft constraints:

- Sooner rather than later: Schedule all meetings as soon as possible.
- A break between meetings: Any two meetings should have at least one time grain break between them.

- Overlapping meetings: To minimize the number of meetings in parallel so people don't have to choose one meeting over the other.
- Assign larger rooms first: If a larger room is available any meeting should be assigned to that room in order to accommodate as many people as possible even if they haven't signed up to that meeting.
- Room stability: If a person has two consecutive meetings with two or less time grains break between them they better be in the same room.

Problem size

> 50meetings-160timegrains-5rooms has 50 meetings, 160 timeGrains and 5 rooms with a search space of 10^145.
> 100meetings-320timegrains-5rooms has 100 meetings, 320 timeGrains and 5 rooms with a search space of 10^320.
> 200meetings-640timegrains-5rooms has 200 meetings, 640 timeGrains and 5 rooms with a search space of 10^701.
> 400meetings-1280timegrains-5rooms has 400 meetings, 1280 timeGrains and 5 rooms with a search space of 10^1522.
> 800meetings-2560timegrains-5rooms has 800 meetings, 2560 timeGrains and 5 rooms with a search space of 10^3285.

4.9. COURSE TIMETABLING (ITC 2007 TRACK 3 - CURRICULUM COURSE SCHEDULING)

Schedule each lecture into a timeslot and into a room.

Hard constraints:

- Teacher conflict: A teacher must not have two lectures in the same period.
- Curriculum conflict: A curriculum must not have two lectures in the same period.
- Room occupancy: Two lectures must not be in the same room in the same period.
- Unavailable period (specified per dataset): A specific lecture must not be assigned to a specific period.

Soft constraints:

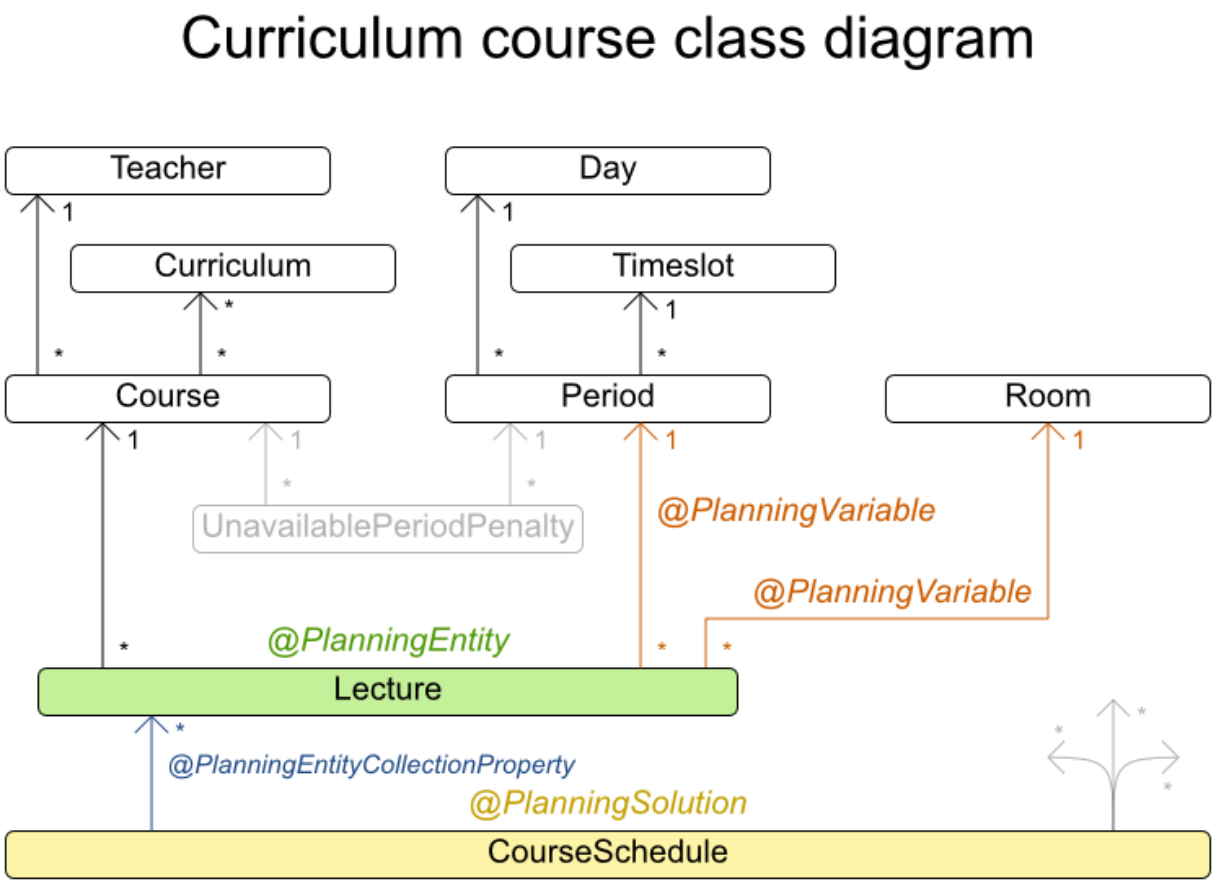
- Room capacity: A room's capacity should not be less than the number of students in its lecture.
- Minimum working days: Lectures of the same course should be spread out into a minimum number of days.
- Curriculum compactness: Lectures belonging to the same curriculum should be adjacent to each other (so in consecutive periods).
- Room stability: Lectures of the same course should be assigned to the same room.

The problem is defined by the International Timetabling Competition 2007 track 3 .

Problem size

```
comp01 has 24 teachers,  14 curricula,  30 courses, 160 lectures, 30 periods,  6 rooms and   53
unavailable period constraints with a search space of  10^360.
comp02 has 71 teachers,  70 curricula,  82 courses, 283 lectures, 25 periods, 16 rooms and  513
unavailable period constraints with a search space of  10^736.
comp03 has 61 teachers,  68 curricula,  72 courses, 251 lectures, 25 periods, 16 rooms and  382
unavailable period constraints with a search space of  10^653.
comp04 has 70 teachers,  57 curricula,  79 courses, 286 lectures, 25 periods, 18 rooms and  396
unavailable period constraints with a search space of  10^758.
comp05 has 47 teachers, 139 curricula,  54 courses, 152 lectures, 36 periods,  9 rooms and  771
unavailable period constraints with a search space of  10^381.
comp06 has 87 teachers,  70 curricula, 108 courses, 361 lectures, 25 periods, 18 rooms and  632
unavailable period constraints with a search space of  10^957.
comp07 has 99 teachers,  77 curricula, 131 courses, 434 lectures, 25 periods, 20 rooms and  667
unavailable period constraints with a search space of 10^1171.
comp08 has 76 teachers,  61 curricula,  86 courses, 324 lectures, 25 periods, 18 rooms and  478
unavailable period constraints with a search space of  10^859.
comp09 has 68 teachers,  75 curricula,  76 courses, 279 lectures, 25 periods, 18 rooms and  405
unavailable period constraints with a search space of  10^740.
comp10 has 88 teachers,  67 curricula, 115 courses, 370 lectures, 25 periods, 18 rooms and  694
unavailable period constraints with a search space of  10^981.
comp11 has 24 teachers,  13 curricula,  30 courses, 162 lectures, 45 periods,  5 rooms and   94
unavailable period constraints with a search space of  10^381.
comp12 has 74 teachers, 150 curricula,  88 courses, 218 lectures, 36 periods, 11 rooms and 1368
unavailable period constraints with a search space of  10^566.
comp13 has 77 teachers,  66 curricula,  82 courses, 308 lectures, 25 periods, 19 rooms and  468
unavailable period constraints with a search space of  10^824.
comp14 has 68 teachers,  60 curricula,  85 courses, 275 lectures, 25 periods, 17 rooms and  486
unavailable period constraints with a search space of  10^722.
```

Figure 4.5. Domain model



4.10. MACHINE REASSIGNMENT (GOOGLE ROADEF 2012)

Assign each process to a machine. All processes already have an original (unoptimized) assignment. Each process requires an amount of each resource (such as CPU or RAM). This is a more complex version of the Cloud Balancing example.

Hard constraints:

- Maximum capacity: The maximum capacity for each resource for each machine must not be exceeded.
- Conflict: Processes of the same service must run on distinct machines.
- Spread: Processes of the same service must be spread out across locations.
- Dependency: The processes of a service depending on another service must run in the neighborhood of a process of the other service.
- Transient usage: Some resources are transient and count towards the maximum capacity of both the original machine as the newly assigned machine.

Soft constraints:

- Load: The safety capacity for each resource for each machine should not be exceeded.

- Balance: Leave room for future assignments by balancing the available resources on each machine.
- Process move cost: A process has a move cost.
- Service move cost: A service has a move cost.
- Machine move cost: Moving a process from machine A to machine B has another A-B specific move cost.

The problem is defined by the Google ROADEF/EURO Challenge 2012 .

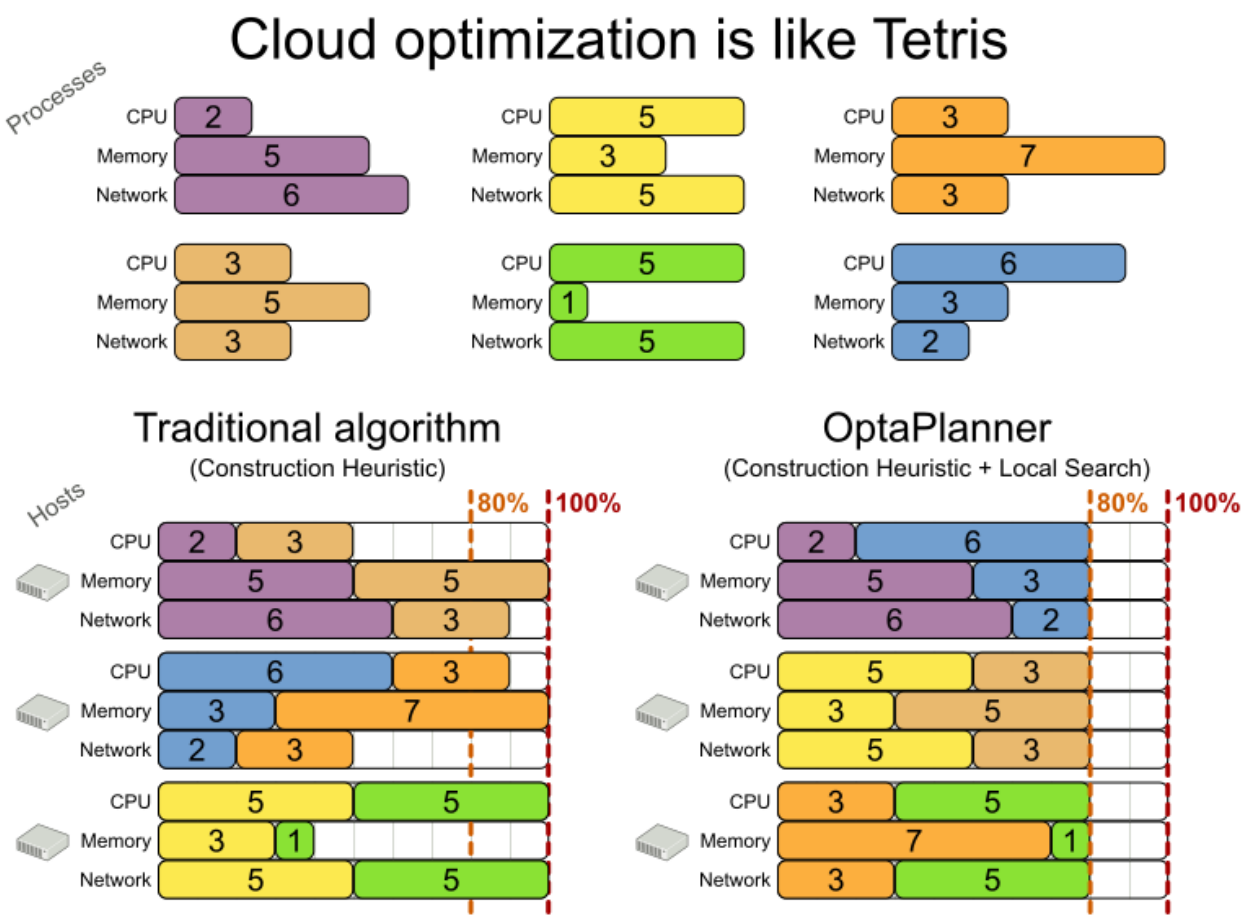
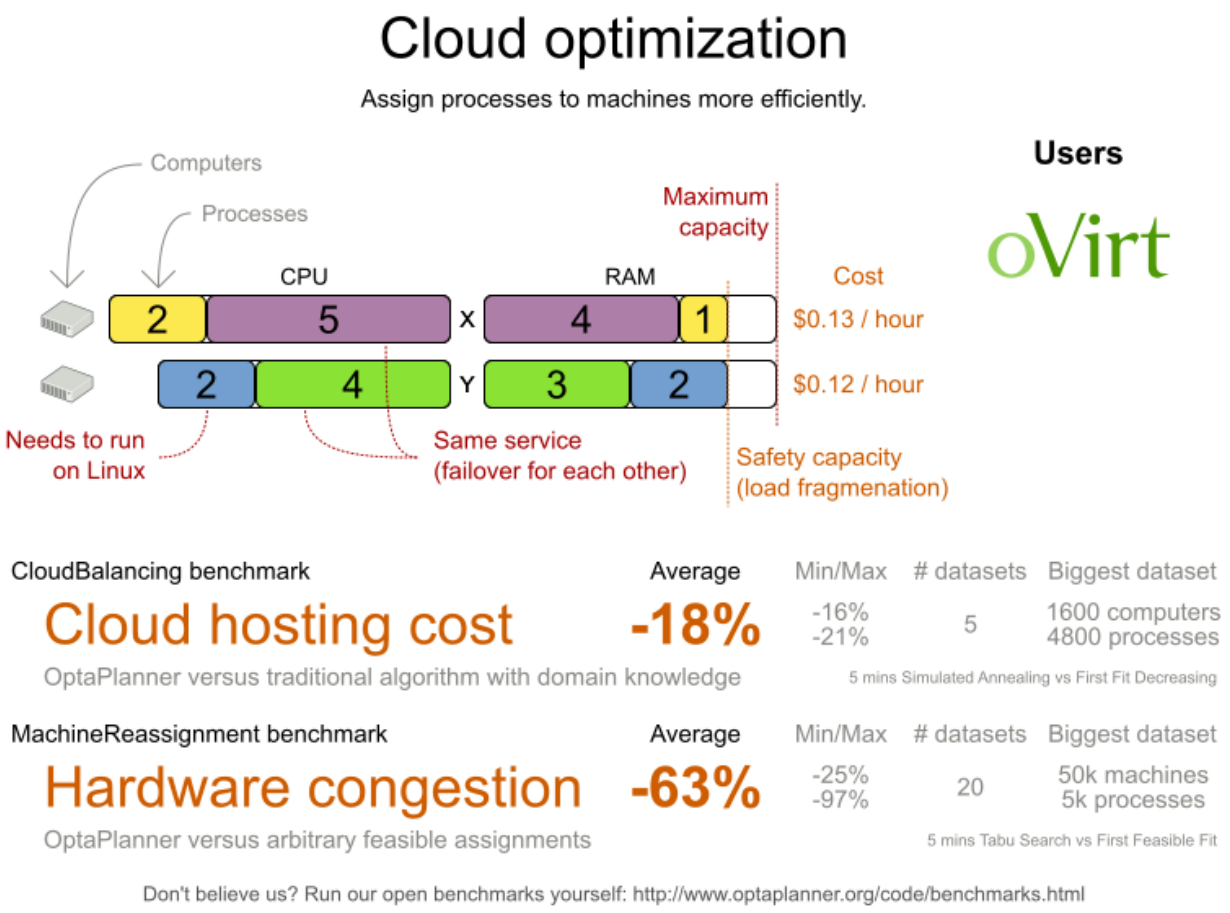


Figure 4.6. Value proposition

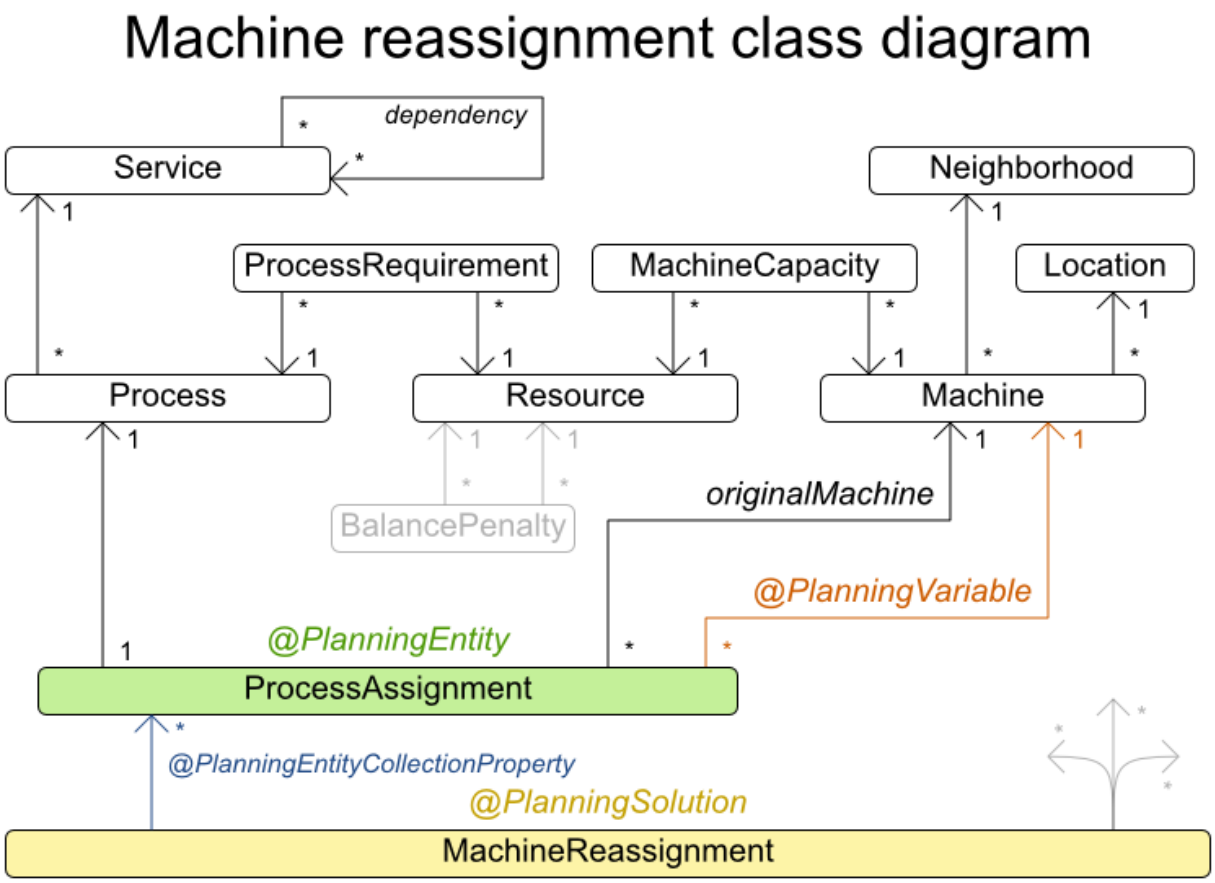


Problem size

```
model_a1_1 has  2 resources,  1 neighborhoods,   4 locations,    4 machines,    79 services,   100
processes and 1 balancePenalties with a search space of     10^60.
model_a1_2 has  4 resources,  2 neighborhoods,   4 locations,  100 machines,   980 services,  1000
processes and 0 balancePenalties with a search space of   10^2000.
model_a1_3 has  3 resources,  5 neighborhoods,  25 locations,  100 machines,   216 services,  1000
processes and 0 balancePenalties with a search space of   10^2000.
model_a1_4 has  3 resources, 50 neighborhoods,  50 locations,   50 machines,   142 services,  1000
processes and 1 balancePenalties with a search space of   10^1698.
model_a1_5 has  4 resources,  2 neighborhoods,   4 locations,   12 machines,   981 services,  1000
processes and 1 balancePenalties with a search space of   10^1079.
model_a2_1 has  3 resources,  1 neighborhoods,   1 locations,  100 machines,  1000 services,  1000
processes and 0 balancePenalties with a search space of   10^2000.
model_a2_2 has 12 resources,  5 neighborhoods,  25 locations,  100 machines,   170 services,  1000
processes and 0 balancePenalties with a search space of   10^2000.
model_a2_3 has 12 resources,  5 neighborhoods,  25 locations,  100 machines,   129 services,  1000
processes and 0 balancePenalties with a search space of   10^2000.
model_a2_4 has 12 resources,  5 neighborhoods,  25 locations,   50 machines,   180 services,  1000
processes and 1 balancePenalties with a search space of   10^1698.
model_a2_5 has 12 resources,  5 neighborhoods,  25 locations,   50 machines,   153 services,  1000
processes and 0 balancePenalties with a search space of   10^1698.
model_b_1  has 12 resources,  5 neighborhoods,  10 locations,  100 machines,  2512 services,  5000
processes and 0 balancePenalties with a search space of  10^10000.
model_b_2  has 12 resources,  5 neighborhoods,  10 locations,  100 machines,  2462 services,  5000
processes and 1 balancePenalties with a search space of  10^10000.
```

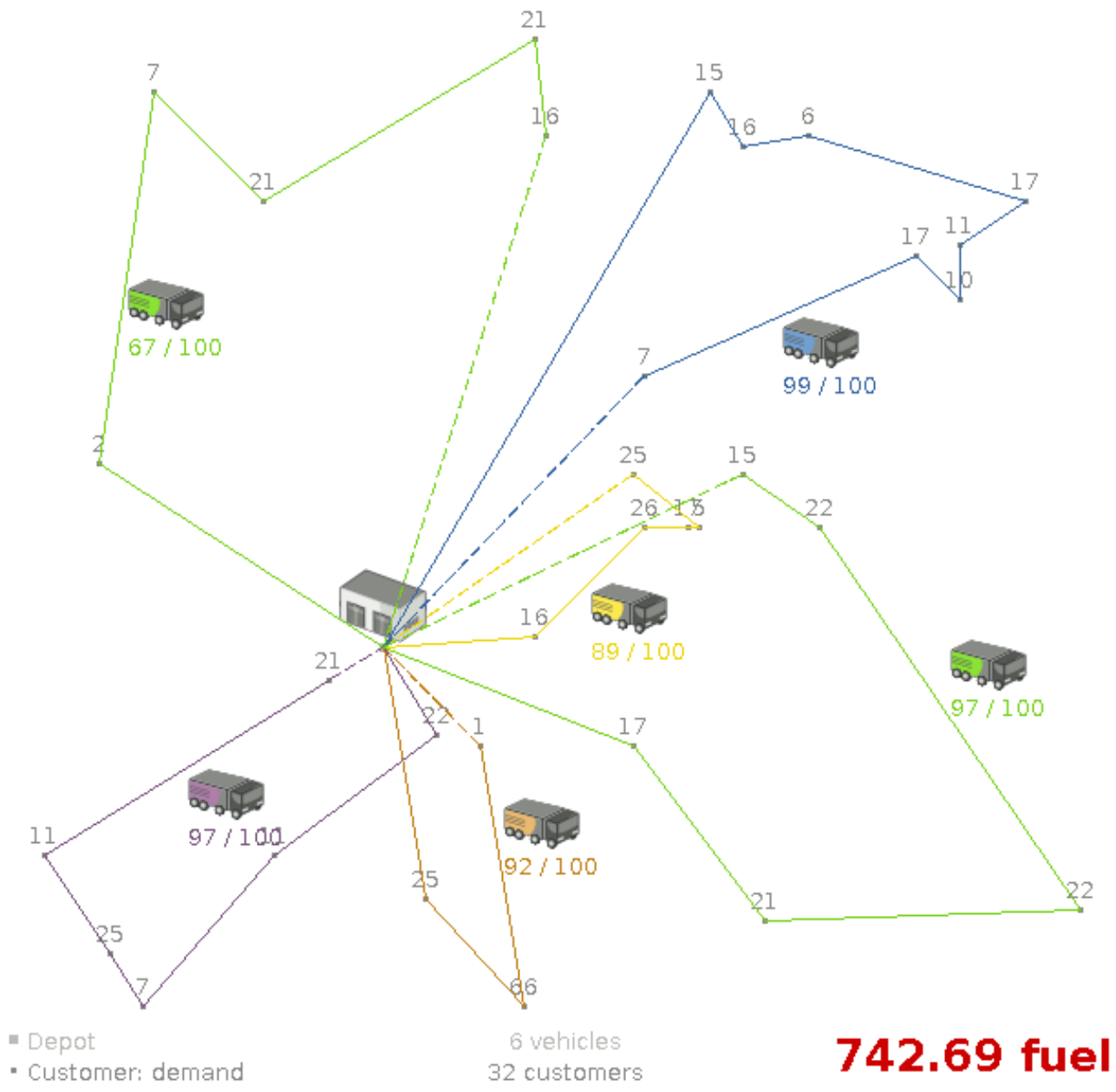
```
model_b_3  has  6 resources,  5 neighborhoods,  10 locations,  100 machines, 15025 services, 20000
processes and 0 balancePenalties with a search space of  10^40000.
model_b_4  has  6 resources,  5 neighborhoods,  50 locations,  500 machines,  1732 services, 20000
processes and 1 balancePenalties with a search space of  10^53979.
model_b_5  has  6 resources,  5 neighborhoods,  10 locations,  100 machines, 35082 services, 40000
processes and 0 balancePenalties with a search space of  10^80000.
model_b_6  has  6 resources,  5 neighborhoods,  50 locations,  200 machines, 14680 services, 40000
processes and 1 balancePenalties with a search space of  10^92041.
model_b_7  has  6 resources,  5 neighborhoods,  50 locations, 4000 machines, 15050 services,
40000 processes and 1 balancePenalties with a search space of 10^144082.
model_b_8  has  3 resources,  5 neighborhoods,  10 locations,  100 machines, 45030 services, 50000
processes and 0 balancePenalties with a search space of 10^100000.
model_b_9  has  3 resources,  5 neighborhoods, 100 locations, 1000 machines,  4609 services,
50000 processes and 1 balancePenalties with a search space of 10^150000.
model_b_10 has  3 resources,  5 neighborhoods, 100 locations, 5000 machines,  4896 services,
50000 processes and 1 balancePenalties with a search space of 10^184948.
```

Figure 4.7. Domain model



4.11. VEHICLE ROUTING

Using a fleet of vehicles, pick up the objects of each customer and bring them to the depot. Each vehicle can service multiple customers, but it has a limited capacity.



Besides the basic case (CVRP), there is also a variant with time windows (CVRPTW).

Hard constraints:

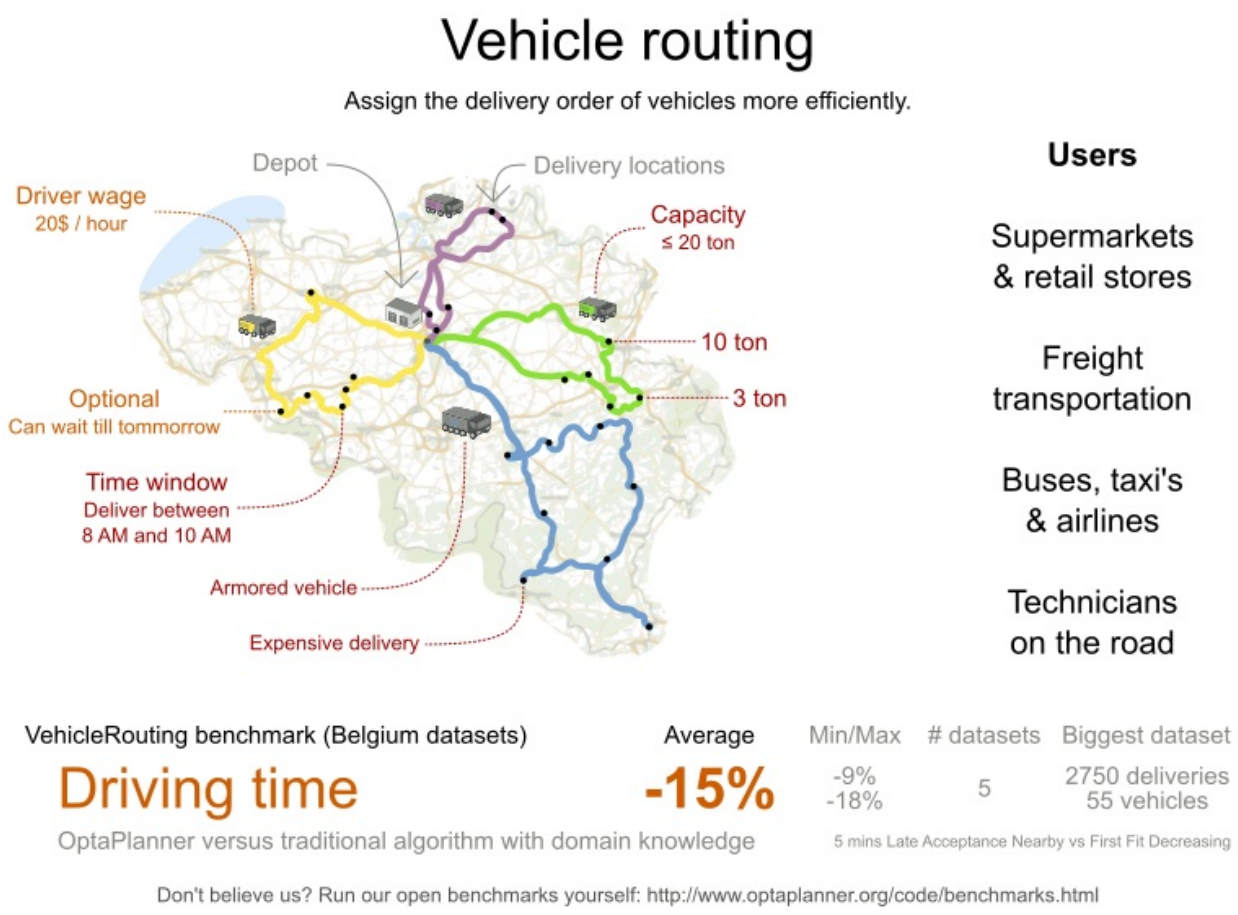
- Vehicle capacity: a vehicle cannot carry more items then its capacity.
- Time windows (only in CVRPTW):
  - Travel time: Traveling from one location to another takes time.
  - Customer service duration: a vehicle must stay at the customer for the length of the service duration.
  - Customer ready time: a vehicle may arrive before the customer's ready time, but it must wait until the ready time before servicing.
  - Customer due time: a vehicle must arrive on time, before the customer's due time.

Soft constraints:

- Total distance: minimize the total distance driven (fuel consumption) of all vehicles.

The capacitated vehicle routing problem (CVRP) and its timewindowed variant (CVRPTW) are defined by the VRP web.

Figure 4.8. Value proposition



Problem size

CVRP instances (without time windows):

```
belgium-n50-k10          has  1 depots, 10 vehicles and   49 customers with a search space of
10^74.
belgium-n100-k10         has  1 depots, 10 vehicles and   99 customers with a search space of
10^170.
belgium-n500-k20         has  1 depots, 20 vehicles and  499 customers with a search space of
10^1168.
belgium-n1000-k20        has  1 depots, 20 vehicles and  999 customers with a search space of
10^2607.
belgium-n2750-k55        has  1 depots, 55 vehicles and 2749 customers with a search space of
10^8380.
belgium-road-km-n50-k10     has  1 depots, 10 vehicles and   49 customers with a search space of
10^74.
belgium-road-km-n100-k10    has  1 depots, 10 vehicles and   99 customers with a search space of
10^170.
belgium-road-km-n500-k20    has  1 depots, 20 vehicles and  499 customers with a search space of
10^1168.
belgium-road-km-n1000-k20   has  1 depots, 20 vehicles and  999 customers with a search space of
10^2607.
belgium-road-km-n2750-k55   has  1 depots, 55 vehicles and 2749 customers with a search space of
```

```
10^8380.
belgium-road-time-n50-k10   has  1 depots, 10 vehicles and   49 customers with a search space of
10^74.
belgium-road-time-n100-k10  has  1 depots, 10 vehicles and   99 customers with a search space of
10^170.
belgium-road-time-n500-k20  has  1 depots, 20 vehicles and  499 customers with a search space of
10^1168.
belgium-road-time-n1000-k20 has  1 depots, 20 vehicles and  999 customers with a search space of
10^2607.
belgium-road-time-n2750-k55 has  1 depots, 55 vehicles and 2749 customers with a search space of
10^8380.
belgium-d2-n50-k10          has  2 depots, 10 vehicles and   48 customers with a search space of
10^74.
belgium-d3-n100-k10         has  3 depots, 10 vehicles and   97 customers with a search space of
10^170.
belgium-d5-n500-k20         has  5 depots, 20 vehicles and  495 customers with a search space of
10^1168.
belgium-d8-n1000-k20        has  8 depots, 20 vehicles and  992 customers with a search space of
10^2607.
belgium-d10-n2750-k55       has 10 depots, 55 vehicles and 2740 customers with a search space of
10^8380.

A-n32-k5  has 1 depots,  5 vehicles and  31 customers with a search space of  10^40.
A-n33-k5  has 1 depots,  5 vehicles and  32 customers with a search space of  10^41.
A-n33-k6  has 1 depots,  6 vehicles and  32 customers with a search space of  10^42.
A-n34-k5  has 1 depots,  5 vehicles and  33 customers with a search space of  10^43.
A-n36-k5  has 1 depots,  5 vehicles and  35 customers with a search space of  10^46.
A-n37-k5  has 1 depots,  5 vehicles and  36 customers with a search space of  10^48.
A-n37-k6  has 1 depots,  6 vehicles and  36 customers with a search space of  10^49.
A-n38-k5  has 1 depots,  5 vehicles and  37 customers with a search space of  10^49.
A-n39-k5  has 1 depots,  5 vehicles and  38 customers with a search space of  10^51.
A-n39-k6  has 1 depots,  6 vehicles and  38 customers with a search space of  10^52.
A-n44-k7  has 1 depots,  7 vehicles and  43 customers with a search space of  10^61.
A-n45-k6  has 1 depots,  6 vehicles and  44 customers with a search space of  10^62.
A-n45-k7  has 1 depots,  7 vehicles and  44 customers with a search space of  10^63.
A-n46-k7  has 1 depots,  7 vehicles and  45 customers with a search space of  10^65.
A-n48-k7  has 1 depots,  7 vehicles and  47 customers with a search space of  10^68.
A-n53-k7  has 1 depots,  7 vehicles and  52 customers with a search space of  10^77.
A-n54-k7  has 1 depots,  7 vehicles and  53 customers with a search space of  10^79.
A-n55-k9  has 1 depots,  9 vehicles and  54 customers with a search space of  10^82.
A-n60-k9  has 1 depots,  9 vehicles and  59 customers with a search space of  10^91.
A-n61-k9  has 1 depots,  9 vehicles and  60 customers with a search space of  10^93.
A-n62-k8  has 1 depots,  8 vehicles and  61 customers with a search space of  10^94.
A-n63-k9  has 1 depots,  9 vehicles and  62 customers with a search space of  10^97.
A-n63-k10 has 1 depots, 10 vehicles and  62 customers with a search space of  10^98.
A-n64-k9  has 1 depots,  9 vehicles and  63 customers with a search space of  10^99.
A-n65-k9  has 1 depots,  9 vehicles and  64 customers with a search space of 10^101.
A-n69-k9  has 1 depots,  9 vehicles and  68 customers with a search space of 10^108.
A-n80-k10 has 1 depots, 10 vehicles and  79 customers with a search space of 10^130.
F-n45-k4  has 1 depots,  4 vehicles and  44 customers with a search space of  10^60.
F-n72-k4  has 1 depots,  4 vehicles and  71 customers with a search space of 10^108.
F-n135-k7 has 1 depots,  7 vehicles and 134 customers with a search space of 10^240.
```

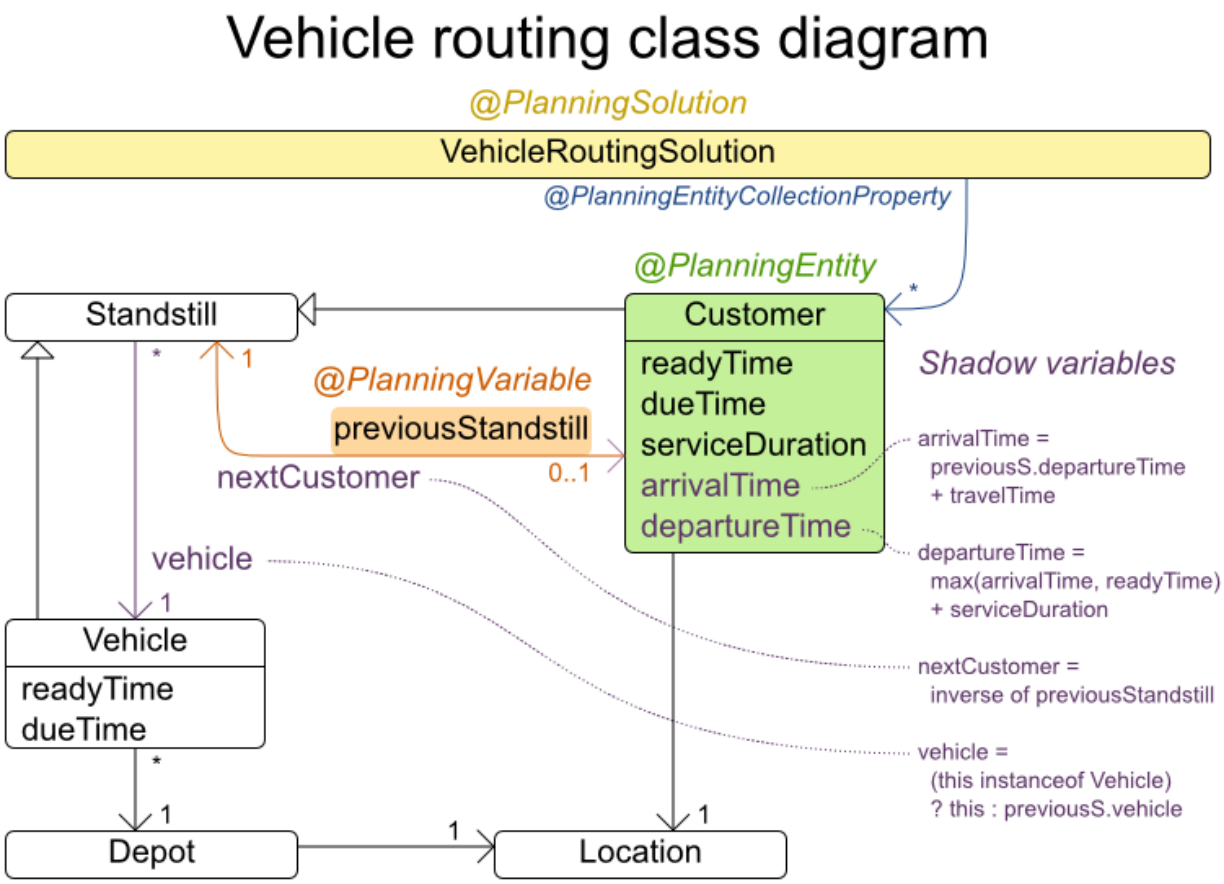
CVRPTW instances (with time windows):

```
belgium-tw-d2-n50-k10    has  2 depots, 10 vehicles and   48 customers with a search space of
```

```
10^74.
belgium-tw-d3-n100-k10   has  3 depots, 10 vehicles and   97 customers with a search space of
10^170.
belgium-tw-d5-n500-k20   has  5 depots, 20 vehicles and  495 customers with a search space of
10^1168.
belgium-tw-d8-n1000-k20  has  8 depots, 20 vehicles and  992 customers with a search space of
10^2607.
belgium-tw-d10-n2750-k55 has 10 depots, 55 vehicles and 2740 customers with a search space of
10^8380.
belgium-tw-n50-k10       has  1 depots, 10 vehicles and   49 customers with a search space of
10^74.
belgium-tw-n100-k10      has  1 depots, 10 vehicles and   99 customers with a search space of
10^170.
belgium-tw-n500-k20      has  1 depots, 20 vehicles and  499 customers with a search space of
10^1168.
belgium-tw-n1000-k20     has  1 depots, 20 vehicles and  999 customers with a search space of
10^2607.
belgium-tw-n2750-k55     has  1 depots, 55 vehicles and 2749 customers with a search space of
10^8380.

Solomon_025_C101       has 1 depots,  25 vehicles and   25 customers with a search space of
10^40.
Solomon_025_C201       has 1 depots,  25 vehicles and   25 customers with a search space of
10^40.
Solomon_025_R101       has 1 depots,  25 vehicles and   25 customers with a search space of
10^40.
Solomon_025_R201       has 1 depots,  25 vehicles and   25 customers with a search space of
10^40.
Solomon_025_RC101      has 1 depots,  25 vehicles and   25 customers with a search space of
10^40.
Solomon_025_RC201      has 1 depots,  25 vehicles and   25 customers with a search space of
10^40.
Solomon_100_C101       has 1 depots,  25 vehicles and  100 customers with a search space of
10^185.
Solomon_100_C201       has 1 depots,  25 vehicles and  100 customers with a search space of
10^185.
Solomon_100_R101       has 1 depots,  25 vehicles and  100 customers with a search space of
10^185.
Solomon_100_R201       has 1 depots,  25 vehicles and  100 customers with a search space of
10^185.
Solomon_100_RC101      has 1 depots,  25 vehicles and  100 customers with a search space of
10^185.
Solomon_100_RC201      has 1 depots,  25 vehicles and  100 customers with a search space of
10^185.
Homberger_0200_C1_2_1  has 1 depots,  50 vehicles and  200 customers with a search space of
10^429.
Homberger_0200_C2_2_1  has 1 depots,  50 vehicles and  200 customers with a search space of
10^429.
Homberger_0200_R1_2_1  has 1 depots,  50 vehicles and  200 customers with a search space of
10^429.
Homberger_0200_R2_2_1  has 1 depots,  50 vehicles and  200 customers with a search space of
10^429.
Homberger_0200_RC1_2_1 has 1 depots,  50 vehicles and  200 customers with a search space of
10^429.
Homberger_0200_RC2_2_1 has 1 depots,  50 vehicles and  200 customers with a search space of
10^429.
```

```
Homberger_0400_C1_4_1  has 1 depots, 100 vehicles and  400 customers with a search space of
10^978.
Homberger_0400_C2_4_1  has 1 depots, 100 vehicles and  400 customers with a search space of
10^978.
Homberger_0400_R1_4_1  has 1 depots, 100 vehicles and  400 customers with a search space of
10^978.
Homberger_0400_R2_4_1  has 1 depots, 100 vehicles and  400 customers with a search space of
10^978.
Homberger_0400_RC1_4_1 has 1 depots, 100 vehicles and  400 customers with a search space of
10^978.
Homberger_0400_RC2_4_1 has 1 depots, 100 vehicles and  400 customers with a search space of
10^978.
Homberger_0600_C1_6_1  has 1 depots, 150 vehicles and  600 customers with a search space of
10^1571.
Homberger_0600_C2_6_1  has 1 depots, 150 vehicles and  600 customers with a search space of
10^1571.
Homberger_0600_R1_6_1  has 1 depots, 150 vehicles and  600 customers with a search space of
10^1571.
Homberger_0600_R2_6_1  has 1 depots, 150 vehicles and  600 customers with a search space of
10^1571.
Homberger_0600_RC1_6_1 has 1 depots, 150 vehicles and  600 customers with a search space of
10^1571.
Homberger_0600_RC2_6_1 has 1 depots, 150 vehicles and  600 customers with a search space of
10^1571.
Homberger_0800_C1_8_1  has 1 depots, 200 vehicles and  800 customers with a search space of
10^2195.
Homberger_0800_C2_8_1  has 1 depots, 200 vehicles and  800 customers with a search space of
10^2195.
Homberger_0800_R1_8_1  has 1 depots, 200 vehicles and  800 customers with a search space of
10^2195.
Homberger_0800_R2_8_1  has 1 depots, 200 vehicles and  800 customers with a search space of
10^2195.
Homberger_0800_RC1_8_1 has 1 depots, 200 vehicles and  800 customers with a search space of
10^2195.
Homberger_0800_RC2_8_1 has 1 depots, 200 vehicles and  800 customers with a search space of
10^2195.
Homberger_1000_C110_1  has 1 depots, 250 vehicles and 1000 customers with a search space of
10^2840.
Homberger_1000_C210_1  has 1 depots, 250 vehicles and 1000 customers with a search space of
10^2840.
Homberger_1000_R110_1  has 1 depots, 250 vehicles and 1000 customers with a search space of
10^2840.
Homberger_1000_R210_1  has 1 depots, 250 vehicles and 1000 customers with a search space of
10^2840.
Homberger_1000_RC110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of
10^2840.
Homberger_1000_RC210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of
10^2840.
```

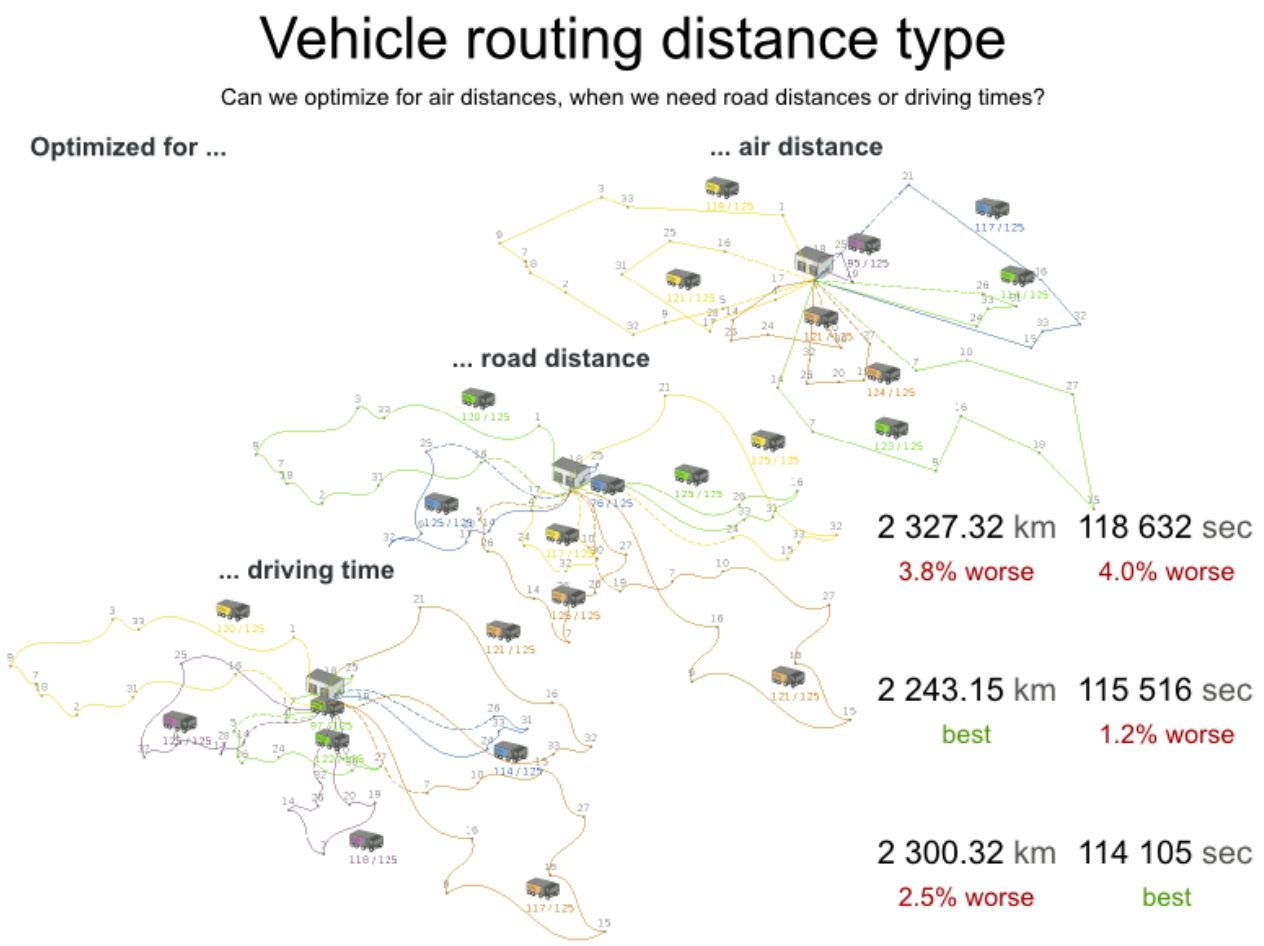
4.11.1. Domain model for Vehicle routing



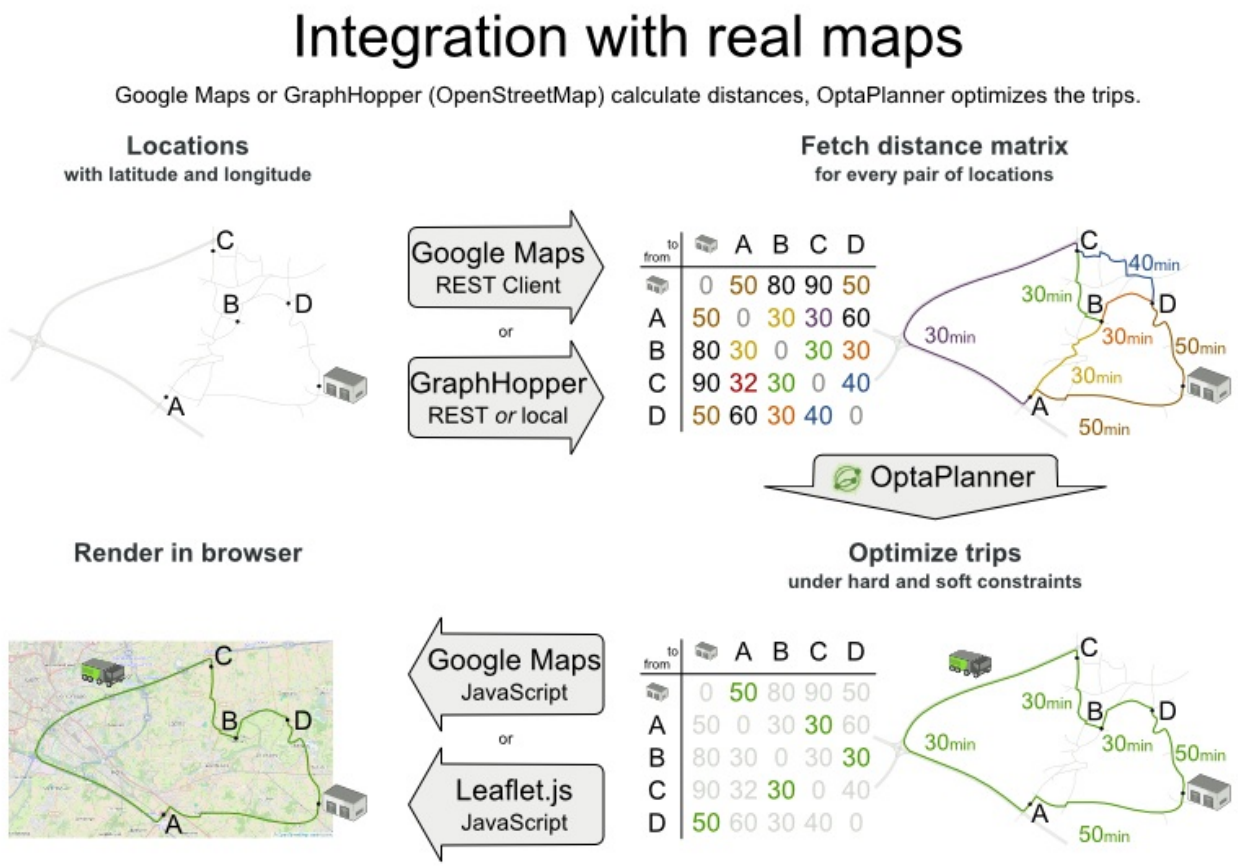
The vehicle routing with timewindows domain model makes heavily use of the shadow variable feature. This allows it to express its constraints more naturally, because properties such as **arrivalTime** and **departureTime**, are directly available on the domain model.

Road Distances Instead of Air Distances

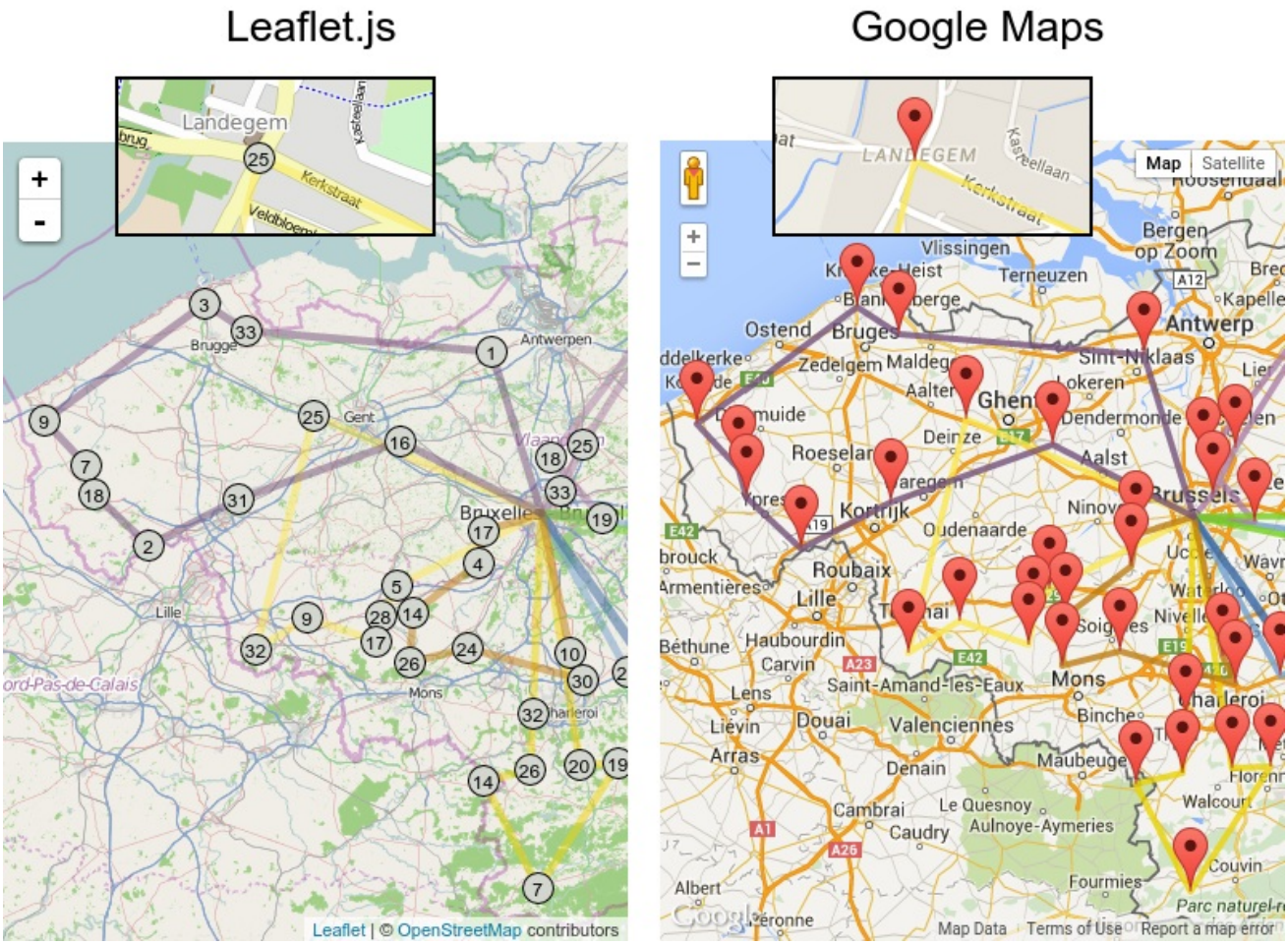
In the real world, vehicles cannot follow a straight line from location to location: they have to use roads and highways. From a business point of view, this matters a lot:



For the optimization algorithm, this does not matter much, as long as the distance between two points can be looked up (and are preferably precalculated). The road cost does not even need to be a distance, it can also be travel time, fuel cost, or a weighted function of those. There are several technologies available to precalculate road costs, such as GraphHopper (embeddable, offline Java engine), Open MapQuest (web service) and Google Maps Client API (web service).

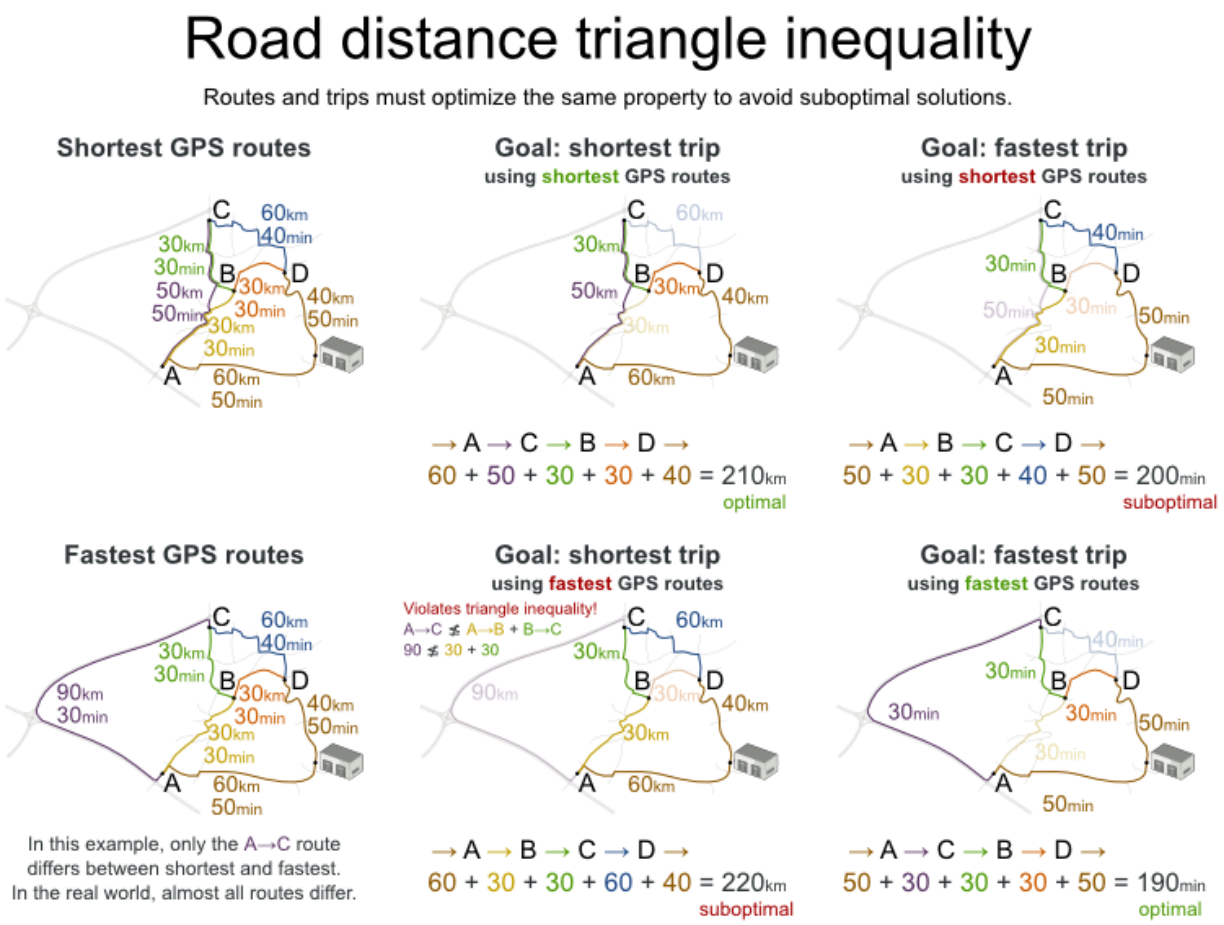


There are also several technologies to render it, such as Leaflet and Google Maps for developers: the **optaplanner-webexamples-*.war** has an example which demonstrates such rendering:



It is even possible to render the actual road routes with GraphHopper or Google Map Directions, but because of route overlaps on highways, it can become harder to see the standstill order:

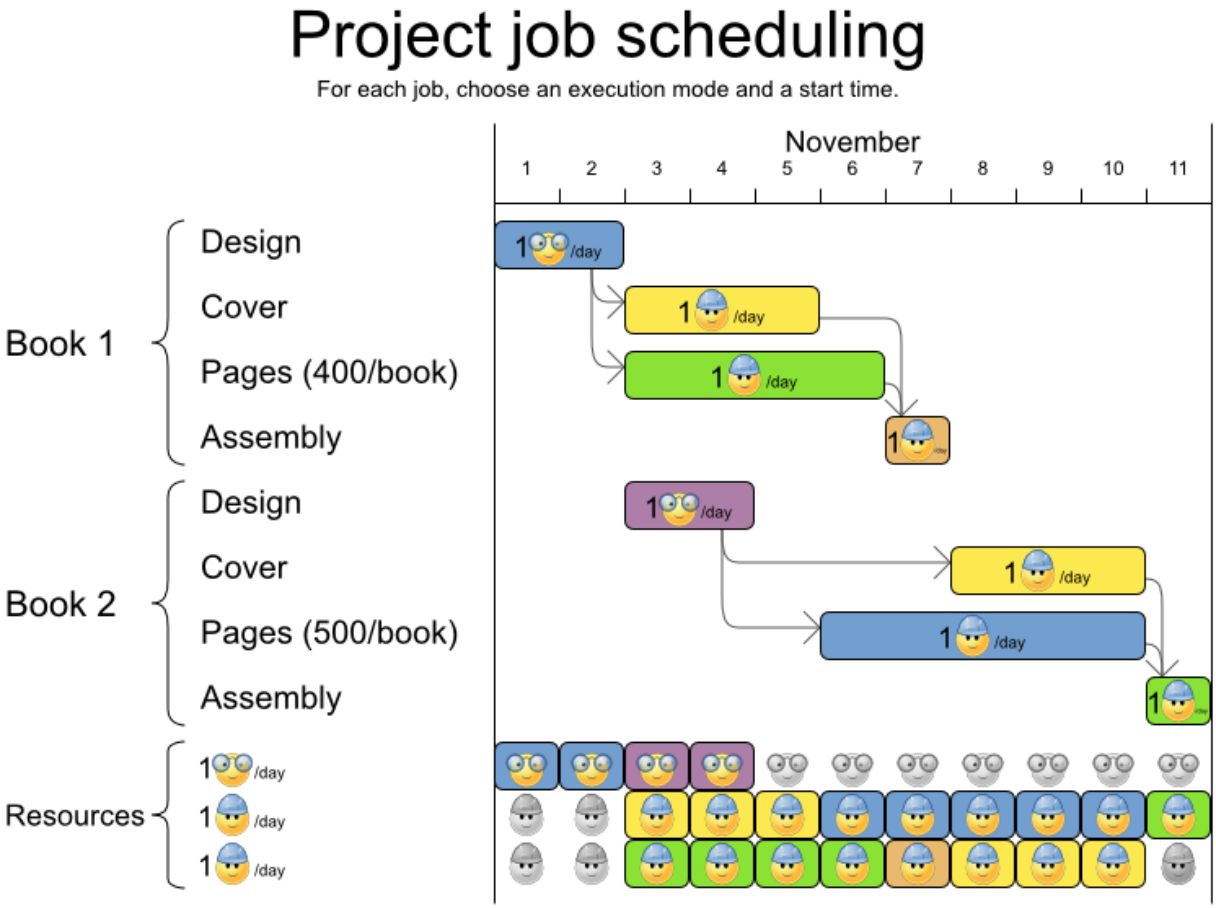
Take special care that the road costs between two points use the same optimization criteria as the one used in Planner. For example, GraphHopper etc will by default return the fastest route, not the shortest route. Don't use the km (or miles) distances of the fastest GPS routes to optimize the shortest trip in Planner: this leads to a suboptimal solution as shown below:



Contrary to popular belief, most users do not want the shortest route: they want the fastest route instead. They prefer highways over normal roads. They prefer normal roads over dirt roads. In the real world, the fastest and shortest route are rarely the same.

4.12. PROJECT JOB SCHEDULING

Schedule all jobs in time and execution mode to minimize project delays. Each job is part of a project. A job can be executed in different ways: each way is an execution mode that implies a different duration but also different resource usages. This is a form of flexible *job shop scheduling*.



Hard constraints:

- Job precedence: a job can only start when all its predecessor jobs are finished.
- Resource capacity: do not use more resources than available.
  - Resources are local (shared between jobs of the same project) or global (shared between all jobs)
  - Resource are renewable (capacity available per day) or nonrenewable (capacity available for all days)

Medium constraints:

- Total project delay: minimize the duration (makespan) of each project.

Soft constraints:

- Total makespan: minimize the duration of the whole multi-project schedule.

The problem is defined by the MISTA 2013 challenge.

Problem size

> Schedule A-1 has 2 projects, 24 jobs, 64 execution modes, 7 resources and 150 resource requirements.
> Schedule A-2 has 2 projects, 44 jobs, 124 execution modes, 7 resources and 420 resource requirements.

```
Schedule A-3  has  2 projects,  64 jobs,  184 execution modes,  7 resources and  630 resource
requirements.
Schedule A-4  has  5 projects,  60 jobs,  160 execution modes, 16 resources and  390 resource
requirements.
Schedule A-5  has  5 projects, 110 jobs,  310 execution modes, 16 resources and  900 resource
requirements.
Schedule A-6  has  5 projects, 160 jobs,  460 execution modes, 16 resources and 1440 resource
requirements.
Schedule A-7  has 10 projects, 120 jobs,  320 execution modes, 22 resources and  900 resource
requirements.
Schedule A-8  has 10 projects, 220 jobs,  620 execution modes, 22 resources and 1860 resource
requirements.
Schedule A-9  has 10 projects, 320 jobs,  920 execution modes, 31 resources and 2880 resource
requirements.
Schedule A-10 has 10 projects, 320 jobs,  920 execution modes, 31 resources and 2970 resource
requirements.
Schedule B-1  has 10 projects, 120 jobs,  320 execution modes, 31 resources and  900 resource
requirements.
Schedule B-2  has 10 projects, 220 jobs,  620 execution modes, 22 resources and 1740 resource
requirements.
Schedule B-3  has 10 projects, 320 jobs,  920 execution modes, 31 resources and 3060 resource
requirements.
Schedule B-4  has 15 projects, 180 jobs,  480 execution modes, 46 resources and 1530 resource
requirements.
Schedule B-5  has 15 projects, 330 jobs,  930 execution modes, 46 resources and 2760 resource
requirements.
Schedule B-6  has 15 projects, 480 jobs, 1380 execution modes, 46 resources and 4500 resource
requirements.
Schedule B-7  has 20 projects, 240 jobs,  640 execution modes, 61 resources and 1710 resource
requirements.
Schedule B-8  has 20 projects, 440 jobs, 1240 execution modes, 42 resources and 3180 resource
requirements.
Schedule B-9  has 20 projects, 640 jobs, 1840 execution modes, 61 resources and 5940 resource
requirements.
Schedule B-10 has 20 projects, 460 jobs, 1300 execution modes, 42 resources and 4260 resource
requirements.
```

4.13. TASK ASSIGNING

Assign each task to a spot in an employee's queue. Each task has a duration which is affected by the employee's affinity level with the task's customer.

Hard constraints:

- Skill: Each task requires one or more skills. The employee must posses all these skills.

Soft level 0 constraints:

- Critical tasks: Complete critical tasks first, sooner than major and minor tasks.

Soft level 1 constraints:

- Minimize makespan: Reduce the time to complete all tasks.
    - Start with the longest working employee first, then the second longest working employee and so forth, to create fairness and load balancing.

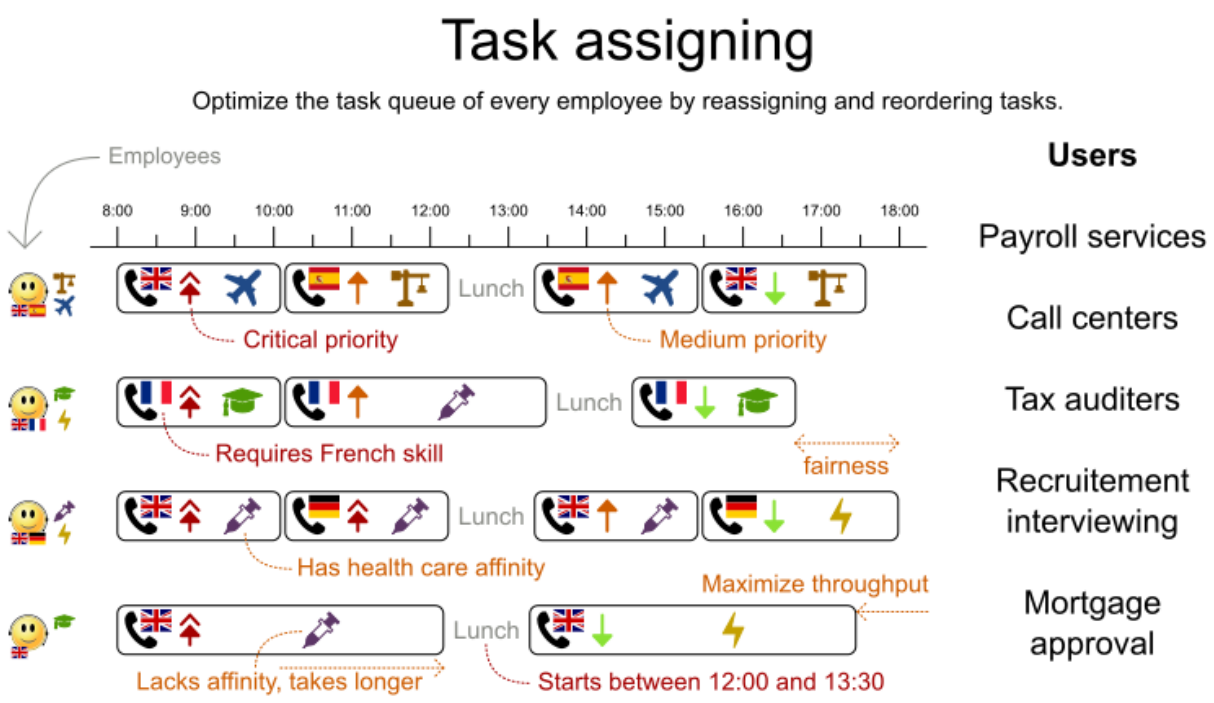
Soft level 2 constraints:

- Major tasks: Complete major tasks as soon as possible, sooner than minor tasks.

Soft level 3 constraints:

- Minor tasks: Complete minor tasks as soon as possible.

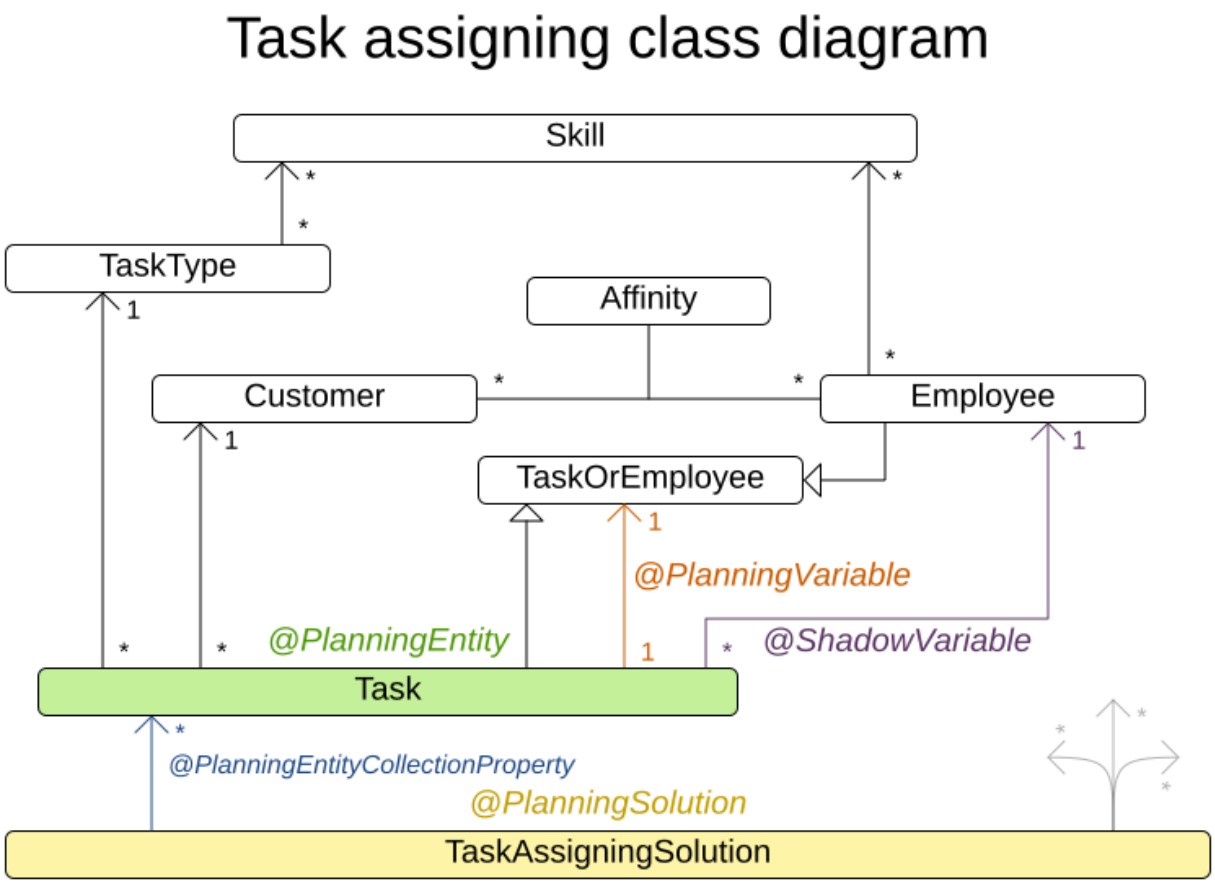
Figure 4.9. Value proposition



Problem size

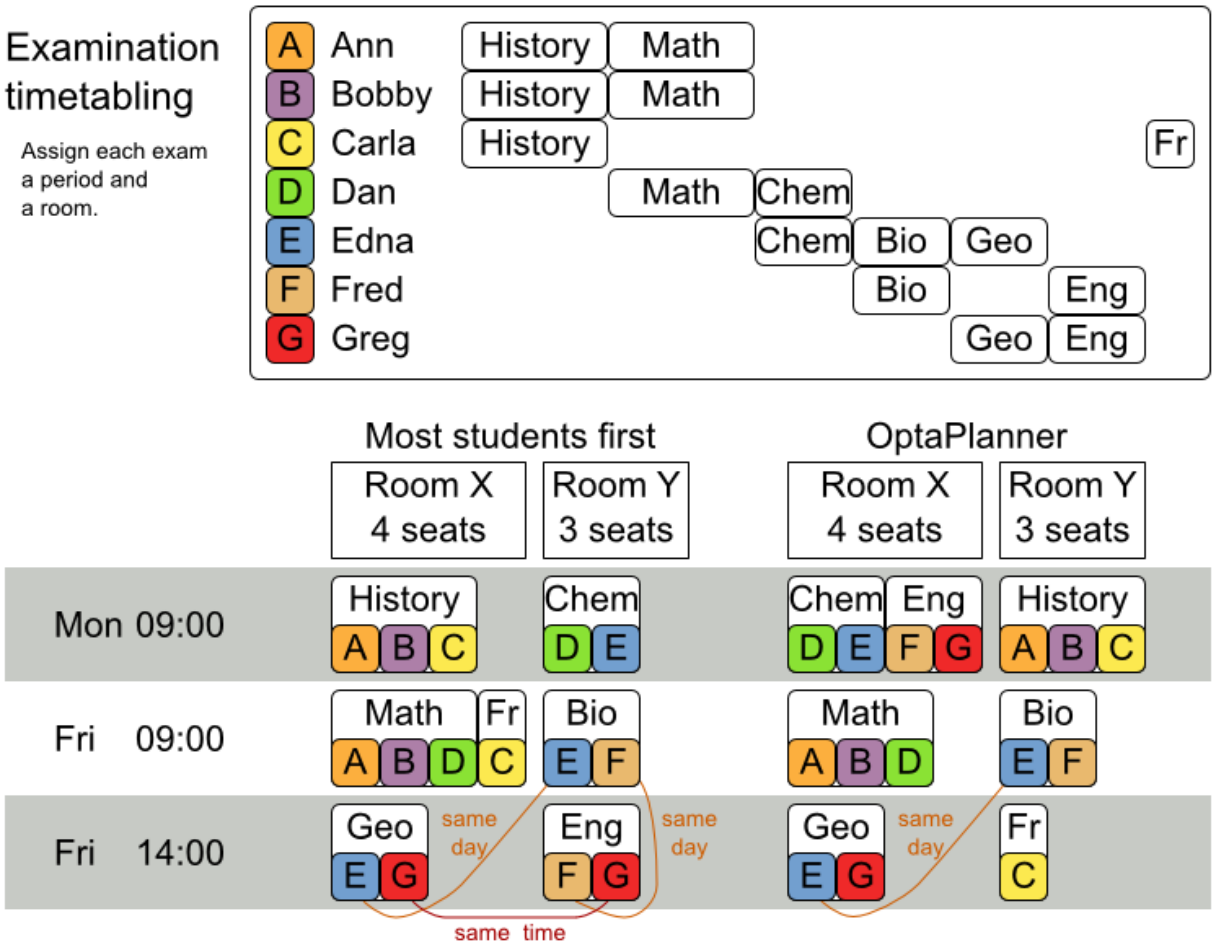
> 24tasks-8employees has 24 tasks, 6 skills, 8 employees, 4 task types and 4 customers with a search space of 10^30.
> 50tasks-5employees has 50 tasks, 5 skills, 5 employees, 10 task types and 10 customers with a search space of 10^69.
> 100tasks-5employees has 100 tasks, 5 skills, 5 employees, 20 task types and 15 customers with a search space of 10^164.
> 500tasks-20employees has 500 tasks, 6 skills, 20 employees, 100 task types and 60 customers with a search space of 10^1168.

Figure 4.10. Domain model



4.14. EXAM TIMETABLING (ITC 2007 TRACK 1 - EXAMINATION)

Schedule each exam into a period and into a room. Multiple exams can share the same room during the same period.



Hard constraints:

- Exam conflict: Two exams that share students must not occur in the same period.
- Room capacity: A room's seating capacity must suffice at all times.
- Period duration: A period's duration must suffice for all of its exams.
- Period related hard constraints (specified per dataset):
  - Coincidence: Two specified exams must use the same period (but possibly another room).
  - Exclusion: Two specified exams must not use the same period.
  - After: A specified exam must occur in a period after another specified exam's period.
- Room related hard constraints (specified per dataset):
  - Exclusive: One specified exam should not have to share its room with any other exam.

Soft constraints (each of which has a parametrized penalty):

- The same student should not have two exams in a row.
- The same student should not have two exams on the same day.
- Period spread: Two exams that share students should be a number of periods apart.
- Mixed durations: Two exams that share a room should not have different durations.

- Front load: Large exams should be scheduled earlier in the schedule.
- Period penalty (specified per dataset): Some periods have a penalty when used.
- Room penalty (specified per dataset): Some rooms have a penalty when used.

It uses large test data sets of real-life universities.

The problem is defined by the International Timetabling Competition 2007 track 1. Geoffrey De Smet finished 4th in that competition with a very early version of Planner. Many improvements have been made since then.

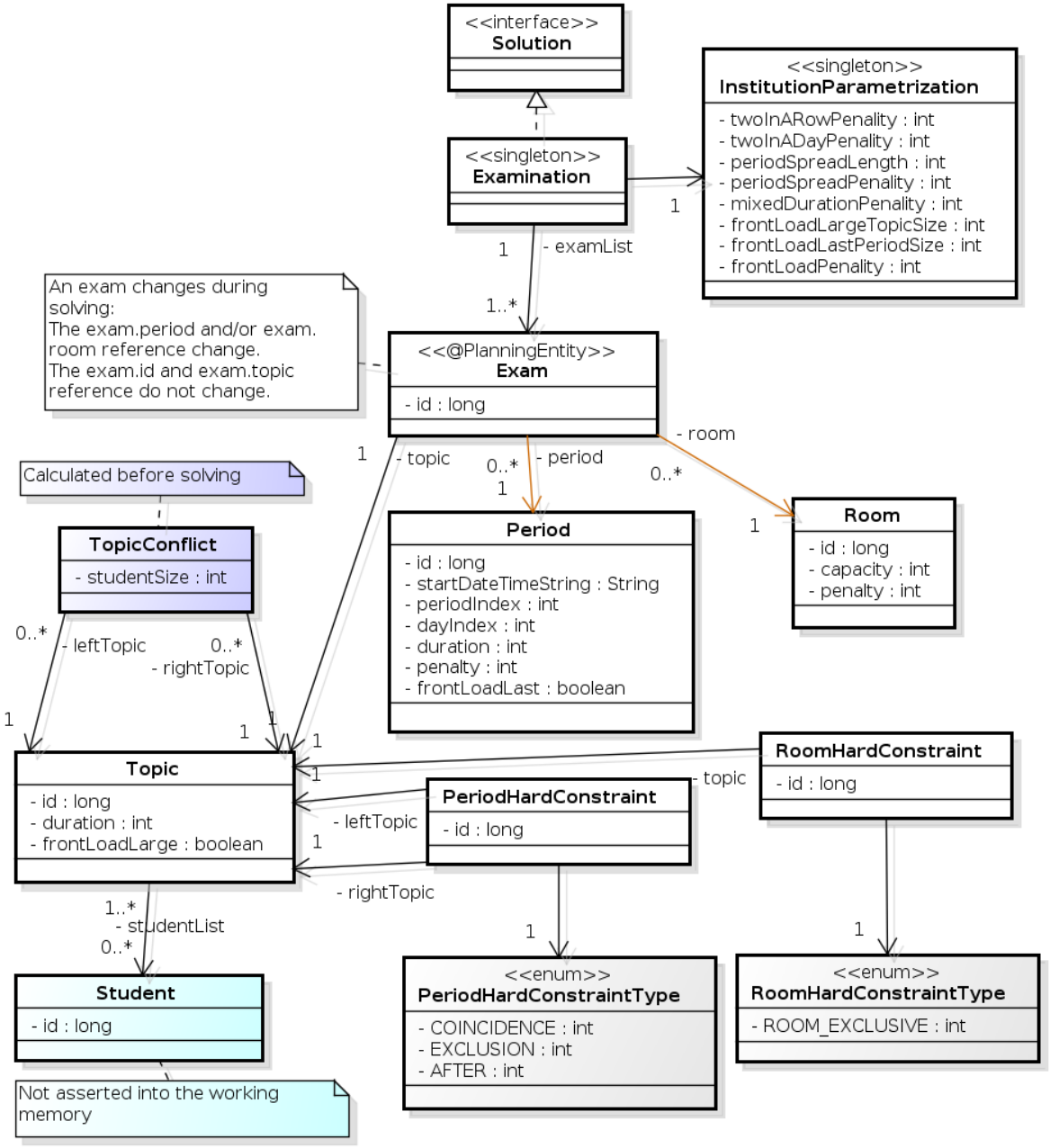
Problem Size

```
exam_comp_set1 has  7883 students,  607 exams, 54 periods,  7 rooms,  12 period constraints and
0 room constraints with a search space of 10^1564.
exam_comp_set2 has 12484 students,  870 exams, 40 periods, 49 rooms,  12 period constraints and
2 room constraints with a search space of 10^2864.
exam_comp_set3 has 16365 students,  934 exams, 36 periods, 48 rooms, 168 period constraints and
15 room constraints with a search space of 10^3023.
exam_comp_set4 has  4421 students,  273 exams, 21 periods,  1 rooms,  40 period constraints and
0 room constraints with a search space of  10^360.
exam_comp_set5 has  8719 students, 1018 exams, 42 periods,  3 rooms,  27 period constraints and
0 room constraints with a search space of 10^2138.
exam_comp_set6 has  7909 students,  242 exams, 16 periods,  8 rooms,  22 period constraints and
0 room constraints with a search space of  10^509.
exam_comp_set7 has 13795 students, 1096 exams, 80 periods, 15 rooms,  28 period constraints and
0 room constraints with a search space of 10^3374.
exam_comp_set8 has  7718 students,  598 exams, 80 periods,  8 rooms,  20 period constraints and
1 room constraints with a search space of 10^1678.
```

4.14.1. Domain model for Exam timetabling

The following diagram shows the main examination domain classes:

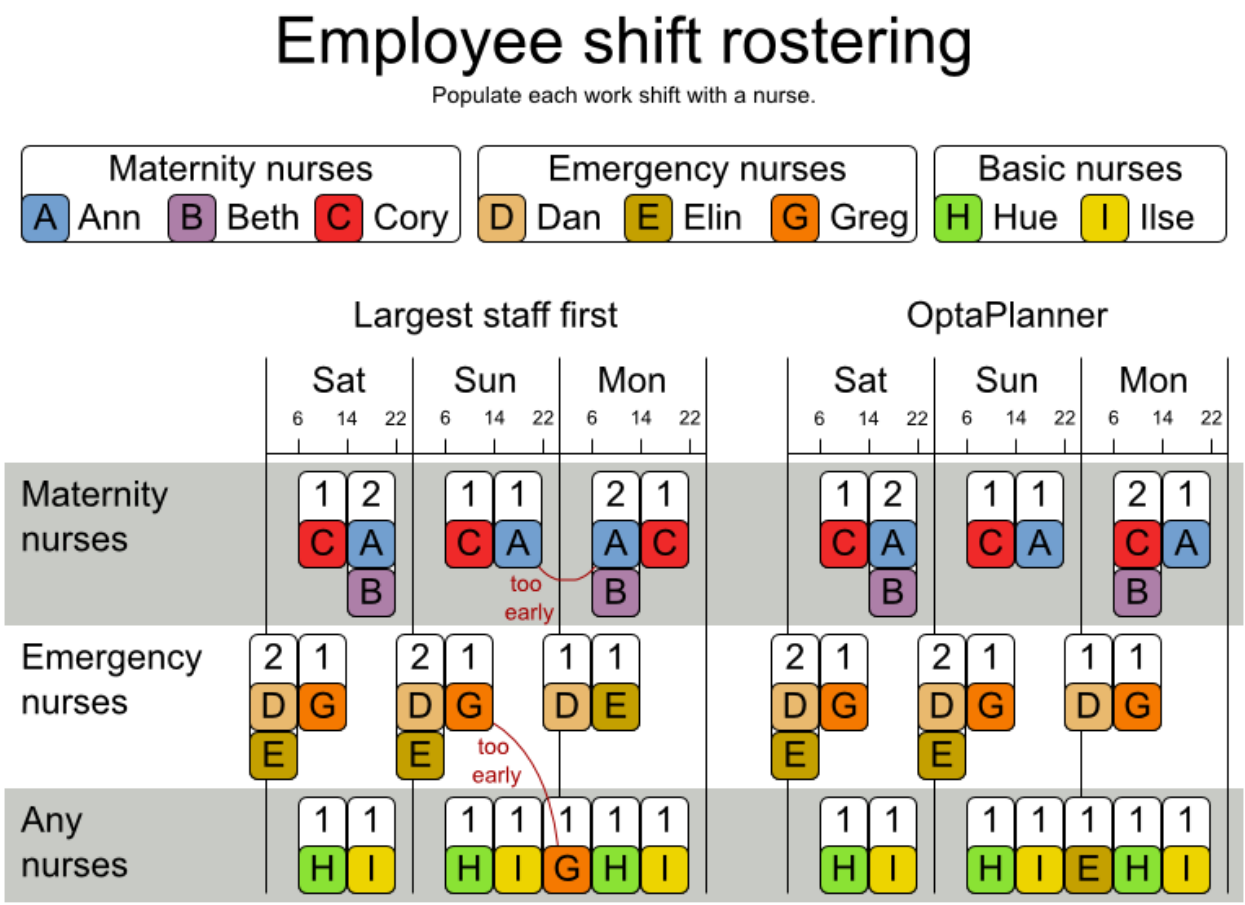
Figure 4.11. Examination domain class diagram



Notice that we've split up the exam concept into an **Exam** class and a **Topic** class. The **Exam** instances change during solving (this is the planning entity class), when their period or room property changes. The **Topic**, **Period** and **Room** instances never change during solving (these are problem facts, just like some other classes).

4.15. NURSE ROSTERING (INRC 2010)

For each shift, assign a nurse to work that shift.



Hard constraints:

- **No unassigned shifts** (built-in): Every shift need to be assigned to an employee.
- **Shift conflict**: An employee can have only one shift per day.

Soft constraints:

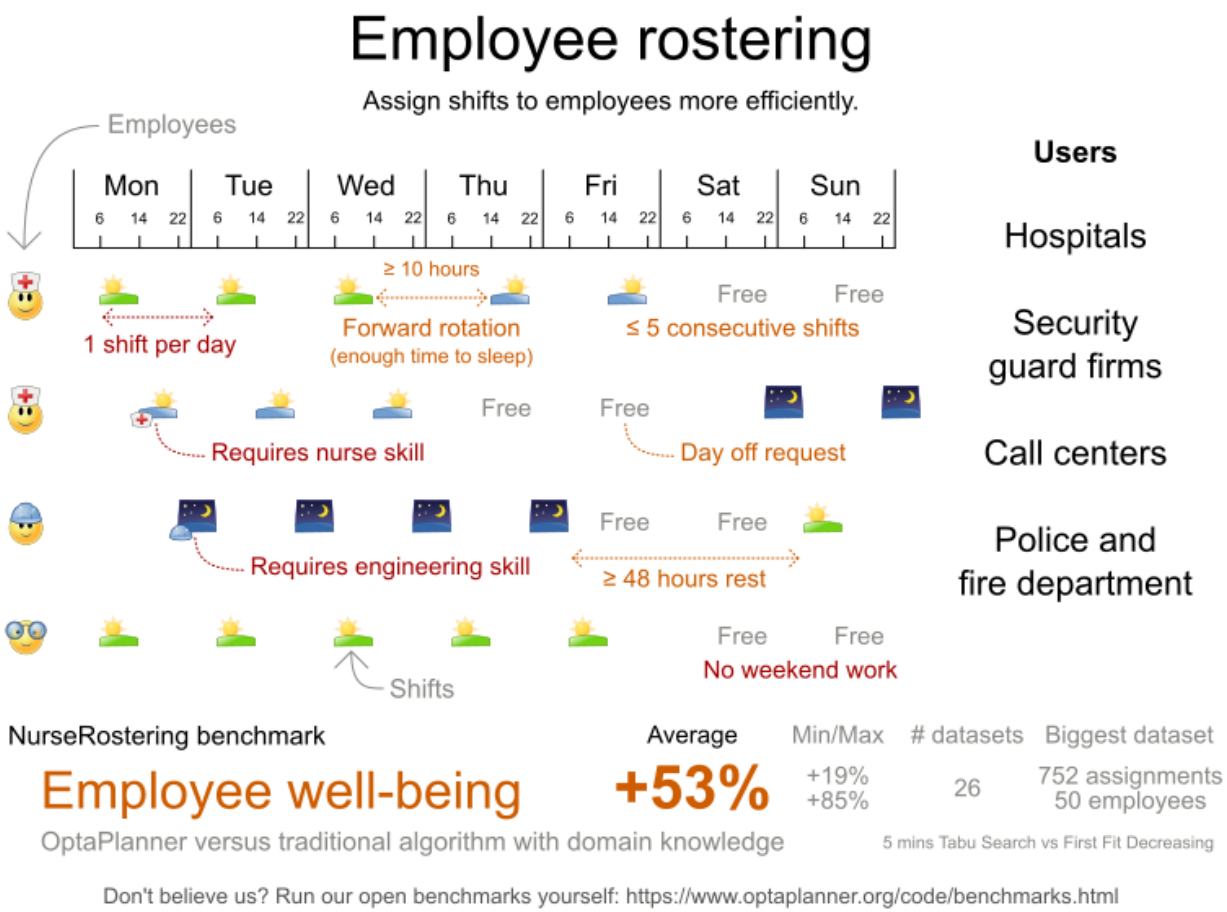
- Contract obligations. The business frequently violates these, so they decided to define these as soft constraints instead of hard constraints.
  - **Minimum and maximum assignments**: Each employee needs to work more than x shifts and less than y shifts (depending on their contract).
  - **Minimum and maximum consecutive working days** Each employee needs to work between x and y days in a row (depending on their contract).
  - **Minimum and maximum consecutive free days** Each employee needs to be free between x and y days in a row (depending on their contract).
  - **Minimum and maximum consecutive working weekends** Each employee needs to work between x and y weekends in a row (depending on their contract).
  - **Complete weekends**: Each employee needs to work every day in a weekend or not at all.
  - **Identical shift types during weekend** Each weekend shift for the same weekend of the same employee must be the same shift type.

  - **Unwanted patterns**: A combination of unwanted shift types in a row. For example: a late shift followed by an early shift followed by a late shift.

- Employee wishes:
  - **Day on request**: An employee wants to work on a specific day.
  - **Day off request**: An employee does not want to work on a specific day.
  - **Shift on request**: An employee wants to be assigned to a specific shift.
  - **Shift off request**: An employee does not want to be assigned to a specific shift.
- **Alternative skill**: An employee assigned to a skill should have a proficiency in every skill required by that shift.

The problem is defined by the International Nurse Rostering Competition 2010 .

Figure 4.12. Value proposition



Problem size

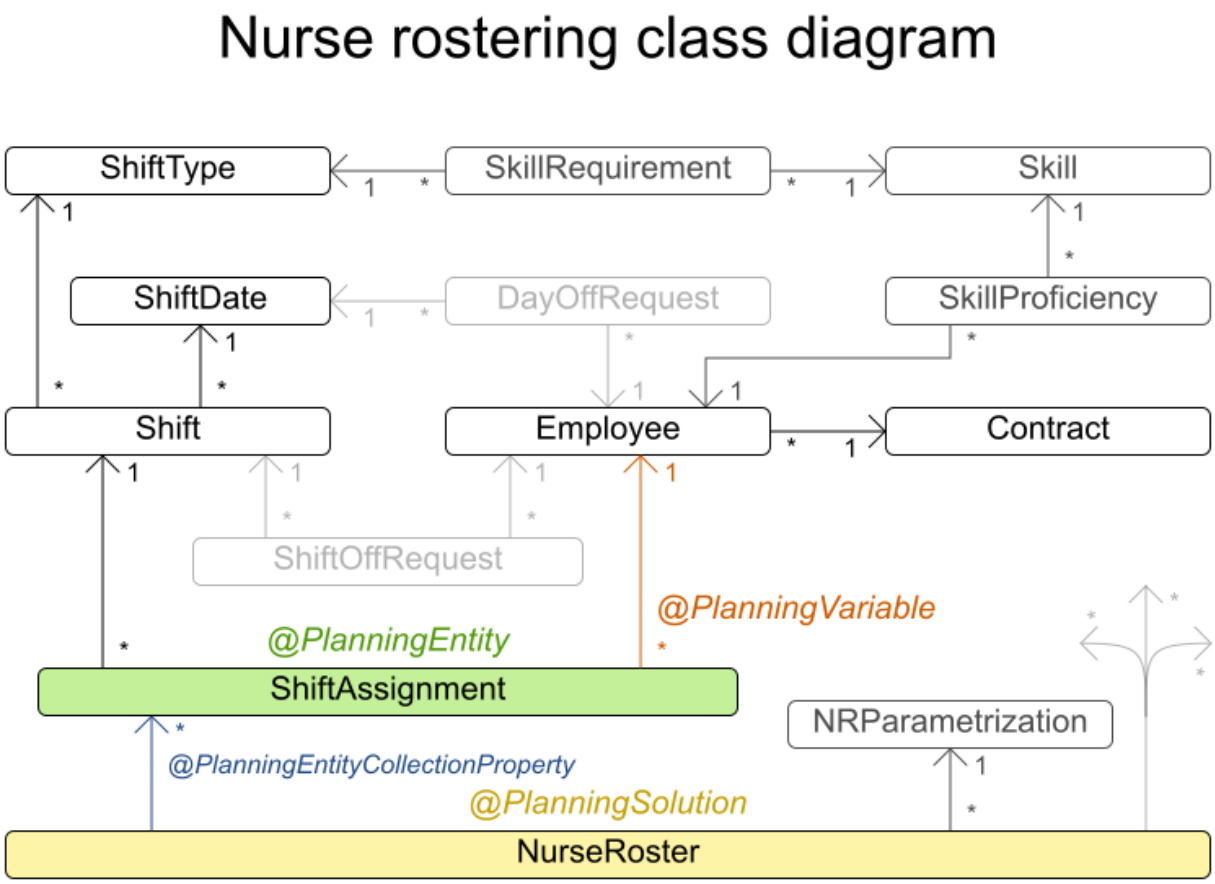
There are three dataset types:

- sprint: must be solved in seconds.
- medium: must be solved in minutes.
- long: must be solved in hours.

```
toy1          has 1 skills, 3 shiftTypes, 2 patterns, 1 contracts,  6 employees,  7 shiftDates,  35
shiftAssignments and   0 requests with a search space of   10^27.
toy2          has 1 skills, 3 shiftTypes, 3 patterns, 2 contracts, 20 employees, 28 shiftDates, 180
shiftAssignments and 140 requests with a search space of  10^234.

sprint01      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint02      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint03      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint04      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint05      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint06      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint07      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint08      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint09      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint10      has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_hint01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_hint02 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_hint03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_late01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_late02 has 1 skills, 3 shiftTypes, 4 patterns, 3 contracts, 10 employees, 28 shiftDates, 144
shiftAssignments and 139 requests with a search space of  10^144.
sprint_late03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160
shiftAssignments and 150 requests with a search space of  10^160.
sprint_late04 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160
shiftAssignments and 150 requests with a search space of  10^160.
sprint_late05 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_late06 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_late07 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.
sprint_late08 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and   0 requests with a search space of  10^152.
sprint_late09 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and   0 requests with a search space of  10^152.
sprint_late10 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152
shiftAssignments and 150 requests with a search space of  10^152.

medium01      has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608
shiftAssignments and 403 requests with a search space of  10^906.
medium02      has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608
shiftAssignments and 403 requests with a search space of  10^906.
```

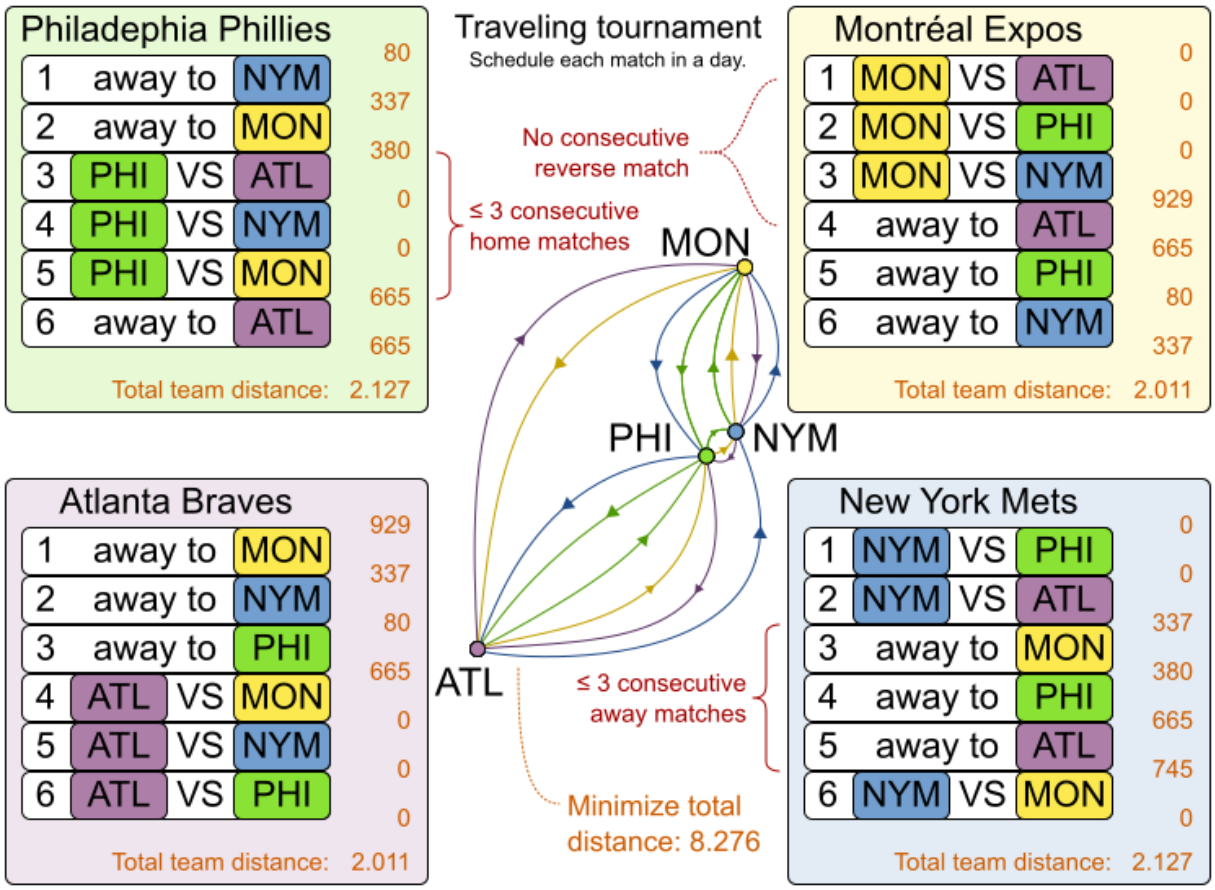
```
medium03      has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608
shiftAssignments and 403 requests with a search space of  10^906.
medium04      has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608
shiftAssignments and 403 requests with a search space of  10^906.
medium05      has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608
shiftAssignments and 403 requests with a search space of  10^906.
medium_hint01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428
shiftAssignments and 390 requests with a search space of  10^632.
medium_hint02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428
shiftAssignments and 390 requests with a search space of  10^632.
medium_hint03 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428
shiftAssignments and 390 requests with a search space of  10^632.
medium_late01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 424
shiftAssignments and 390 requests with a search space of  10^626.
medium_late02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428
shiftAssignments and 390 requests with a search space of  10^632.
medium_late03 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 30 employees, 28 shiftDates, 428
shiftAssignments and 390 requests with a search space of  10^632.
medium_late04 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 416
shiftAssignments and 390 requests with a search space of  10^614.
medium_late05 has 2 skills, 5 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 452
shiftAssignments and 390 requests with a search space of  10^667.

long01        has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740
shiftAssignments and 735 requests with a search space of 10^1250.
long02        has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740
shiftAssignments and 735 requests with a search space of 10^1250.
long03        has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740
shiftAssignments and 735 requests with a search space of 10^1250.
long04        has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740
shiftAssignments and 735 requests with a search space of 10^1250.
long05        has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740
shiftAssignments and 735 requests with a search space of 10^1250.
long_hint01   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740
shiftAssignments and   0 requests with a search space of 10^1257.
long_hint02   has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740
shiftAssignments and   0 requests with a search space of 10^1257.
long_hint03   has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740
shiftAssignments and   0 requests with a search space of 10^1257.
long_late01   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752
shiftAssignments and   0 requests with a search space of 10^1277.
long_late02   has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752
shiftAssignments and   0 requests with a search space of 10^1277.
long_late03   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752
shiftAssignments and   0 requests with a search space of 10^1277.
long_late04   has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752
shiftAssignments and   0 requests with a search space of 10^1277.
long_late05   has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740
shiftAssignments and   0 requests with a search space of 10^1257.
```

Figure 4.13. Domain model



4.16. TRAVELING TOURNAMENT PROBLEM (TTP)

Schedule matches between *n* teams.



Hard constraints:

- Each team plays twice against every other team: once home and once away.
- Each team has exactly one match on each timeslot.
- No team must have more than three consecutive home or three consecutive away matches.
- No repeaters: no two consecutive matches of the same two opposing teams.

Soft constraints:

- Minimize the total distance traveled by all teams.

The problem is defined on Michael Trick's website (which contains the world records too) .

Problem size

```
1-nl04    has  6 days,  4 teams and   12 matches with a search space of    10^5.
1-nl06    has 10 days,  6 teams and   30 matches with a search space of   10^19.
1-nl08    has 14 days,  8 teams and   56 matches with a search space of   10^43.
1-nl10    has 18 days, 10 teams and   90 matches with a search space of   10^79.
1-nl12    has 22 days, 12 teams and  132 matches with a search space of  10^126.
1-nl14    has 26 days, 14 teams and  182 matches with a search space of  10^186.
1-nl16    has 30 days, 16 teams and  240 matches with a search space of  10^259.
2-bra24   has 46 days, 24 teams and  552 matches with a search space of  10^692.
3-nfl16   has 30 days, 16 teams and  240 matches with a search space of  10^259.
3-nfl18   has 34 days, 18 teams and  306 matches with a search space of  10^346.
```

```
3-nfl20    has 38 days, 20 teams and  380 matches with a search space of  10^447.
3-nfl22    has 42 days, 22 teams and  462 matches with a search space of  10^562.
3-nfl24    has 46 days, 24 teams and  552 matches with a search space of  10^692.
3-nfl26    has 50 days, 26 teams and  650 matches with a search space of  10^838.
3-nfl28    has 54 days, 28 teams and  756 matches with a search space of  10^999.
3-nfl30    has 58 days, 30 teams and  870 matches with a search space of 10^1175.
3-nfl32    has 62 days, 32 teams and  992 matches with a search space of 10^1367.
4-super04  has  6 days,  4 teams and   12 matches with a search space of    10^5.
4-super06  has 10 days,  6 teams and   30 matches with a search space of   10^19.
4-super08  has 14 days,  8 teams and   56 matches with a search space of   10^43.
4-super10  has 18 days, 10 teams and   90 matches with a search space of   10^79.
4-super12  has 22 days, 12 teams and  132 matches with a search space of  10^126.
4-super14  has 26 days, 14 teams and  182 matches with a search space of  10^186.
5-galaxy04 has  6 days,  4 teams and   12 matches with a search space of    10^5.
5-galaxy06 has 10 days,  6 teams and   30 matches with a search space of   10^19.
5-galaxy08 has 14 days,  8 teams and   56 matches with a search space of   10^43.
5-galaxy10 has 18 days, 10 teams and   90 matches with a search space of   10^79.
5-galaxy12 has 22 days, 12 teams and  132 matches with a search space of  10^126.
5-galaxy14 has 26 days, 14 teams and  182 matches with a search space of  10^186.
5-galaxy16 has 30 days, 16 teams and  240 matches with a search space of  10^259.
5-galaxy18 has 34 days, 18 teams and  306 matches with a search space of  10^346.
5-galaxy20 has 38 days, 20 teams and  380 matches with a search space of  10^447.
5-galaxy22 has 42 days, 22 teams and  462 matches with a search space of  10^562.
5-galaxy24 has 46 days, 24 teams and  552 matches with a search space of  10^692.
5-galaxy26 has 50 days, 26 teams and  650 matches with a search space of  10^838.
5-galaxy28 has 54 days, 28 teams and  756 matches with a search space of  10^999.
5-galaxy30 has 58 days, 30 teams and  870 matches with a search space of 10^1175.
5-galaxy32 has 62 days, 32 teams and  992 matches with a search space of 10^1367.
5-galaxy34 has 66 days, 34 teams and 1122 matches with a search space of 10^1576.
5-galaxy36 has 70 days, 36 teams and 1260 matches with a search space of 10^1801.
5-galaxy38 has 74 days, 38 teams and 1406 matches with a search space of 10^2042.
5-galaxy40 has 78 days, 40 teams and 1560 matches with a search space of 10^2301.
```

4.17. CHEAP TIME SCHEDULING

Schedule all tasks in time and on a machine to minimize power cost. Power prices differs in time. This is a form of *job shop scheduling*.

Hard constraints:

- Start time limits: Each task must start between its earliest start and latest start limit.
- Maximum capacity: The maximum capacity for each resource for each machine must not be exceeded.
- Startup and shutdown: Each machine must be active in the periods during which it has assigned tasks. Between tasks it is allowed to be idle to avoid startup and shutdown costs.

Medium constraints:

- Power cost: Minimize the total power cost of the whole schedule.
  - Machine power cost: Each active or idle machine consumes power, which infers a power cost (depending on the power price during that time).

  - Task power cost: Each task consumes power too, which infers a power cost (depending on the power price during its time).
  - Machine startup and shutdown cost: Every time a machine starts up or shuts down, an extra cost is inflicted.

Soft constraints (addendum to the original problem definition):

- Start early: Prefer starting a task sooner rather than later.

The problem is defined by the ICON challenge.

Problem size

```
sample01   has 3 resources,   2 machines, 288 periods and   25 tasks with a search space of
10^53.
sample02   has 3 resources,   2 machines, 288 periods and   50 tasks with a search space of
10^114.
sample03   has 3 resources,   2 machines, 288 periods and  100 tasks with a search space of
10^226.
sample04   has 3 resources,   5 machines, 288 periods and  100 tasks with a search space of
10^266.
sample05   has 3 resources,   2 machines, 288 periods and  250 tasks with a search space of
10^584.
sample06   has 3 resources,   5 machines, 288 periods and  250 tasks with a search space of
10^673.
sample07   has 3 resources,   2 machines, 288 periods and 1000 tasks with a search space of
10^2388.
sample08   has 3 resources,   5 machines, 288 periods and 1000 tasks with a search space of
10^2748.
sample09   has 4 resources,  20 machines, 288 periods and 2000 tasks with a search space of
10^6668.
instance00 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^595.
instance01 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^599.
instance02 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^599.
instance03 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^591.
instance04 has 1 resources,  10 machines, 288 periods and  200 tasks with a search space of
10^590.
instance05 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^667.
instance06 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^660.
instance07 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^662.
instance08 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^651.
instance09 has 2 resources,  25 machines, 288 periods and  200 tasks with a search space of
10^659.
instance10 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1657.
instance11 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1644.
```

```
instance12 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1637.
instance13 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1659.
instance14 has 2 resources,  20 machines, 288 periods and  500 tasks with a search space of
10^1643.
instance15 has 3 resources,  40 machines, 288 periods and  500 tasks with a search space of
10^1782.
instance16 has 3 resources,  40 machines, 288 periods and  500 tasks with a search space of
10^1778.
instance17 has 3 resources,  40 machines, 288 periods and  500 tasks with a search space of
10^1764.
instance18 has 3 resources,  40 machines, 288 periods and  500 tasks with a search space of
10^1769.
instance19 has 3 resources,  40 machines, 288 periods and  500 tasks with a search space of
10^1778.
instance20 has 3 resources,  50 machines, 288 periods and 1000 tasks with a search space of
10^3689.
instance21 has 3 resources,  50 machines, 288 periods and 1000 tasks with a search space of
10^3678.
instance22 has 3 resources,  50 machines, 288 periods and 1000 tasks with a search space of
10^3706.
instance23 has 3 resources,  50 machines, 288 periods and 1000 tasks with a search space of
10^3676.
instance24 has 3 resources,  50 machines, 288 periods and 1000 tasks with a search space of
10^3681.
instance25 has 3 resources,  60 machines, 288 periods and 1000 tasks with a search space of
10^3774.
instance26 has 3 resources,  60 machines, 288 periods and 1000 tasks with a search space of
10^3737.
instance27 has 3 resources,  60 machines, 288 periods and 1000 tasks with a search space of
10^3744.
instance28 has 3 resources,  60 machines, 288 periods and 1000 tasks with a search space of
10^3731.
instance29 has 3 resources,  60 machines, 288 periods and 1000 tasks with a search space of
10^3746.
instance30 has 4 resources,  70 machines, 288 periods and 2000 tasks with a search space of
10^7718.
instance31 has 4 resources,  70 machines, 288 periods and 2000 tasks with a search space of
10^7740.
instance32 has 4 resources,  70 machines, 288 periods and 2000 tasks with a search space of
10^7686.
instance33 has 4 resources,  70 machines, 288 periods and 2000 tasks with a search space of
10^7672.
instance34 has 4 resources,  70 machines, 288 periods and 2000 tasks with a search space of
10^7695.
instance35 has 4 resources,  80 machines, 288 periods and 2000 tasks with a search space of
10^7807.
instance36 has 4 resources,  80 machines, 288 periods and 2000 tasks with a search space of
10^7814.
instance37 has 4 resources,  80 machines, 288 periods and 2000 tasks with a search space of
10^7764.
instance38 has 4 resources,  80 machines, 288 periods and 2000 tasks with a search space of
10^7736.
instance39 has 4 resources,  80 machines, 288 periods and 2000 tasks with a search space of
10^7783.
```

```
instance40 has 4 resources,  90 machines, 288 periods and 4000 tasks with a search space of
10^15976.
instance41 has 4 resources,  90 machines, 288 periods and 4000 tasks with a search space of
10^15935.
instance42 has 4 resources,  90 machines, 288 periods and 4000 tasks with a search space of
10^15887.
instance43 has 4 resources,  90 machines, 288 periods and 4000 tasks with a search space of
10^15896.
instance44 has 4 resources,  90 machines, 288 periods and 4000 tasks with a search space of
10^15885.
instance45 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20173.
instance46 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20132.
instance47 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20126.
instance48 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20110.
instance49 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of
10^20078.
```

4.18. INVESTMENT ASSET CLASS ALLOCATION (PORTFOLIO OPTIMIZATION)

Decide the relative quantity to invest in each asset class.

Hard constraints:

- Risk maximum: the total standard deviation must not be higher than the standard deviation maximum.
  - Total standard deviation calculation takes asset class correlations into account by applying Markowitz Portfolio Theory.
- Region maximum: Each region has a quantity maximum.
- Sector maximum: Each sector has a quantity maximum.

Soft constraints:

- Maximize expected return.

Problem size

```
de_smet_1 has 1 regions, 3 sectors and 11 asset classes with a search space of 10^4.
irrinki_1 has 2 regions, 3 sectors and 6 asset classes with a search space of 10^3.
```

Larger datasets have not been created or tested yet, but should not pose a problem. A good source of data is this Asset Correlation website.

4.19. CONFERENCE SCHEDULING

Assign each conference talk to a timeslot and a room. Timeslots can overlap. Read/write to/from an ***.xlsx** file that can be edited with LibreOffice or Excel too.

Hard constraints:

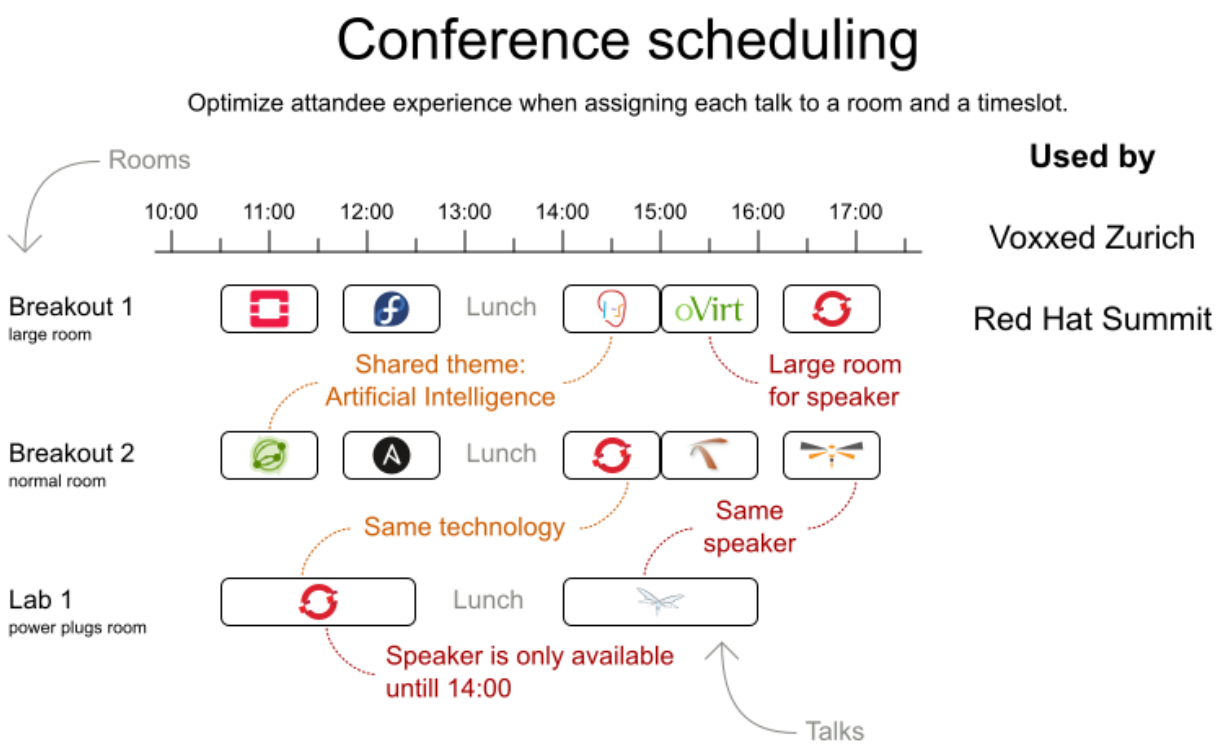
- Talk type of timeslot: The type of a talk must match the timeslot's talk type.
- Room unavailable timeslots: A talk's room must be available during the talk's timeslot.
- Room conflict: Two talks can't use the same room during overlapping timeslots.
- Speaker unavailable timeslots: Every talk's speaker must be available during the talk's timeslot.
- Speaker conflict: Two talks can't share a speaker during overlapping timeslots.
- Generic purpose timeslot and room tags
  - Speaker required timeslot tag: If a speaker has a required timeslot tag, then all his/her talks must be assigned to a timeslot with that tag.
  - Speaker prohibited timeslot tag: If a speaker has a prohibited timeslot tag, then all his/her talks cannot be assigned to a timeslot with that tag.
  - Talk required timeslot tag: If a talk has a required timeslot tag, then it must be assigned to a timeslot with that tag.
  - Talk prohibited timeslot tag: If a talk has a prohibited timeslot tag, then it cannot be assigned to a timeslot with that tag.
  - Speaker required room tag: If a speaker has a required room tag, then all his/her talks must be assigned to a room with that tag.
  - Speaker prohibited room tag: If a speaker has a prohibited room tag, then all his/her talks cannot be assigned to a room with that tag.
  - Talk required room tag: If a talk has a required room tag, then it must be assigned to a room with that tag.
  - Talk prohibited room tag: If a talk has a prohibited room tag, then it cannot be assigned to a room with that tag.
- Talk mutually-exclusive-talks tag: Talks that share such a tag must not be scheduled in overlapping timeslots.
- Talk prerequisite talks: A talk must be scheduled after all its prerequisite talks.

Soft constraints:

- Theme track conflict: Minimize the number of talks that share a same theme tag during overlapping timeslots.
- Sector conflict: Minimize the number of talks that share a same sector tag during overlapping timeslots.
- Content audience level flow violation: For every content tag, schedule the introductory talks before the advanced talks.
- Audience level diversity: For every timeslot, maximize the number of talks with a different audience level.
- Language diversity: For every timeslot, maximize the number of talks with a different language.

- Generic purpose timeslot and room tags
  - Speaker preferred timeslot tag: If a speaker has a preferred timeslot tag, then all his/her talks should be assigned to a timeslot with that tag.
  - Speaker undesired timeslot tag: If a speaker has a undesired timeslot tag, then all his/her talks should not be assigned to a timeslot with that tag.
  - Talk preferred timeslot tag: If a talk has a preferred timeslot tag, then it should be assigned to a timeslot with that tag.
  - Talk undesired timeslot tag: If a talk has a undesired timeslot tag, then it should not be assigned to a timeslot with that tag.
  - Speaker preferred room tag: If a speaker has a preferred room tag, then all his/her talks should be assigned to a room with that tag.
  - Speaker undesired room tag: If a speaker has a undesired room tag, then all his/her talks should not be assigned to a room with that tag.
  - Talk preferred room tag: If a talk has a preferred room tag, then it should be assigned to a room with that tag.
  - Talk undesired room tag: If a talk has a undesired room tag, then it should not be assigned to a room with that tag.
- Same day talks: All talks that share a same theme tag or content tag should be scheduled in the minimum number of days (ideally in the same day).

Figure 4.14. Value proposition



Problem size

> 18talks-6timeslots-5rooms has 18 talks, 6 timeslots and 5 rooms with a search space of 10^26.
> 36talks-12timeslots-5rooms has 36 talks, 12 timeslots and 5 rooms with a search space of 10^64.
> 72talks-12timeslots-10rooms has 72 talks, 12 timeslots and 10 rooms with a search space of 10^149.
> 108talks-18timeslots-10rooms has 108 talks, 18 timeslots and 10 rooms with a search space of 10^243.
> 216talks-18timeslots-20rooms has 216 talks, 18 timeslots and 20 rooms with a search space of 10^552.

4.20. ROCK TOUR

Drive the rock bus from show to show, but schedule shows only on available days.

Hard constraints:

- Schedule every required show.
- Schedule as many shows as possible.

Medium constraints:

- Maximize revenue opportunity.

- Minimize driving time.
- Visit sooner than later.

Soft constraints:

- Avoid long driving times.

Problem size

> 47shows has 47 shows with a search space of 10^59.

4.21. FLIGHT CREW SCHEDULING

Assign flights to pilots and flight attendants.

Hard constraints:

- Required skill: each flight assignment has a required skill. For example, flight AB0001 requires 2 pilots and 3 flight attendants.
- Flight conflict: each employee can only attend one flight at the same time
- Transfer between two flights: between two flights, an employee must be able to transfer from the arrival airport to the departure airport. For example, Ann arrives in Brussels at 10:00 and departs in Amsterdam at 15:00.
- Employee unavailability: the employee must be available on the day of the flight. For example, Ann is on PTO on 1-Feb.

Soft constraints:

- First assignment departing from home
- Last assignment arriving at home
- Load balance flight duration total per employee

Problem size

> 175flights-7days-Europe has 2 skills, 50 airports, 150 employees, 175 flights and 875 flight assignments with a search space of 10^1904.
> 700flights-28days-Europe has 2 skills, 50 airports, 150 employees, 700 flights and 3500 flight assignments with a search space of 10^7616.
> 875flights-7days-Europe has 2 skills, 50 airports, 750 employees, 875 flights and 4375 flight assignments with a search space of 10^12578.
> 175flights-7days-US has 2 skills, 48 airports, 150 employees, 175 flights and 875 flight assignments with a search space of 10^1904.

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Monday, July 19, 2021.